به نام خدا

مجموعه مستندات
پروژه درس سیستم عامل ۲

مسعود مشرف جوادی ۸۲۱۷۳۵۰۱

مسعود آخوندی ۸۵۲۰۰۱۸۵

ابوالحسن شمسایی ۸۶۳۰۰۲۹۸

بهمن ۸۶

فهرست مستندات

نمایه مستندات

مستندات تولید شده برای این پروژه به شرح ذیل می‌باشند. اکیداً پیشنهاد می‌شود برای درک بهتر سیستم، مستندات، به ترتیب آورده شده در زیر مورد مطالعه قرار بگیرند.

نمودارهای مورد کاربرد

نمودارهای مورد کاربرد سیستم در واقع وظیفه‌مندی سیستم را از دید کاربر بیرونی نمایش می‌دهند و جزء اصولی‌ترین و بدوی‌ترین نمودارهای به کار رفته در تحلیل و طراحی بر اساس RUP می‌باشند. نمودارهای مورد کاربرد اصولاً شئ‌گرا نیستند اما در بسیاری از متولوژی‌های تحلیل و طراحی شئ‌گرا به کار رفته و در درک نیازمندی‌ها کمک بسزایی می‌کنند.

نمودار مؤلّفه

در این مستند لیست انواع مؤلّفه ها و نحوه کنارهم قرار گرفتن مؤلّفه های مختلف سیستم و ارتباط‌های هر کدام نشان داده شده است. این نمودار را در آخر فاز طراحی و آغاز فاز پیاده سازی تولید گردیده است.

نمودارهای فعالیت

برای تحقق موردهای کاربرد و در سطحی بالاتر از آن برای مدلسازی هر نوع فرایندی که توالی به نوعی در آن اهمیت دارد از نمودارهای فعالیت استفاده می‌شود. مهمترین کاربرد این نمودارها در مدل کردن فرایندهای تجاری است.

این نمودار در کنار نمودارهای مورد کاربرد، درک بهتری از سیستم به ما می‌دهند. در واقع این نمودارها نقطه ضعف نمودارهای مورد کاربرد را که همان ترتیب انجام فعالیت‌ها در سیستم بود، پوشش داده و نحوه و ترتیب انجام فعالیت‌های درون هر مورد کاربرد را مشخص می‌کند.

پیاده سازی نمودار فعالیت

جهت نمایش ساختار ایستای برنامه و اجزای محقق کننده وظایف کاربرد به کار می رود در این نمودار علاوه بر نمایش ساختار کلّی هر فایل روابط میان فایل‌های مختلف با نمودارهای فعالیت ذکر شده نشان داده شده است. این نمودار در کنار نمودارهای تعاملی جهت نمایش نحوه تحقق وظایف موارد کاربرد به کار می رود. این نمودارها در کنار نمودارهای تعاملی به شکل تکاملی و تکراری ساخته می شوند.

نمودار استقرار

این نمودار نمایشی از نحوه قرار گرفتن مؤلفه های مختلف برروی گره‌های محاسباتی در سیستم را در اختیار می گذارد. این نمودار همراه با نمودار مؤلّفه تولید شده و به بعد از تعیین معماری نرم‌افزار ، در پایان فاز پیاده سازی همراه با ان استفاده می شود.

مورد کاربرد

مقدمه

در این مستند مشخصات موارد کاربرد برای این پروژه که در قالب یک بازی نوشته شده آورده شده است. از آنجائیکه اندازه برنامه چندان بزرگ نیست. کل سیستم در غالب یک زیرسیستم دیده شده است.

شرح بازیگر ها

برای این سیستم دو نوع بازیگر در نظر گرفته شده است.

مدیر سیستم

مدیر سیستم قادر به راه‌اندازی سرور بازی و خاموش کردن آن می باشد. جزئیات بیشتر آن در بخش توضیح موارد کاربرد آورد شده است.

بازیکن

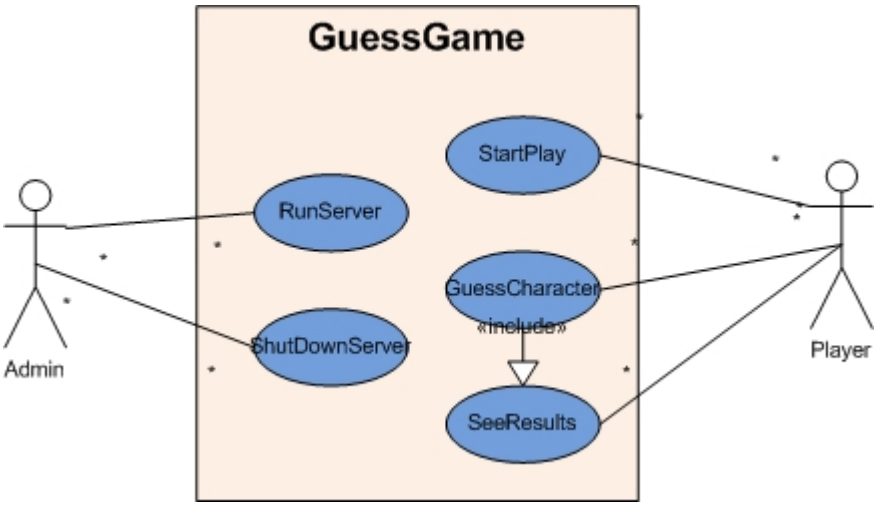
بازیکن فردی است که در بازی شرکت می کند و در واقع استفاده کننده اصلی سیستم است. جزئیات بیشتر آن در بخش توضیح موارد کاربرد آورد شده است.

زیر سیستم اصلی

توصیف:

این زیرسیستم تنها زیرسیستم این برنامه است و کلّ روند بازی را مدیریت می کند.

نمودار مورد کاربرد:



شکل ۱نمودار مورد کاربرد زیرسیستم اصلی

مشخصات هر مورد کاربرد:

در ادامه مشخصات هر مورد کاربرد (مورد کاربرد) آورده شده است:

مورد کاربرد: RunServer
شماره: ۱
توصیف اجمالی: راه اندازی سرور بازی
عامل اصلی: Admin (مدیر سیستم)
عامل فرعی: ندارد
شرایط اولیه: ندارد
روند اصلی: ۱. چک کردن ورودی ها و تنظیمات برنامه ۲. اگر تنظیمات درست است ۲.۱. راه اندازی برنامه سرور بازی ۲.۲. ایجاد ساختار داده های مناسب برای پیوستن مشتری های بازی ۲.۳. منتظر بمان تا مشتری ها به سیستم متّصل شوند ۳. با پیغام مناسب کاربر را مطّلع کن
شرایط نهایی: برنامه سرور بازی اجرا شده و منتظر پیوستن برنامه های مشتری است.
روند جایگزین: ندارد

مورد کاربرد: ShutDownServer
شماره: ۲
توصیف اجمالی: مدیر سیستم سرور بازی را خاموش می کند.
عامل اصلی: Admin (مدیر سیستم)
عامل فرعی: ندارد
شرایط اولیه: سرور بازی در حال اجرا باشد.
روند اصلی: ۱. اگر برنامه بازی مشتری در حال اجرا است ۱.۱. به ازای تمامی برنامه های بازی مشتری پیغام پایان بازی را به آنها بفرست. ۲. از بازی خارج شو.
شرایط نهایی: برنامه سرور بازی از حافظه خارج شده است.
روند جایگزین: ندارد

مورد کاربرد: StartPlay
شماره: ۳
توصیف اجمالی: برنامه بازی مشتری شروع به کار می کند و به برنامه سرور بازی متّصل می گردد.
عامل اصلی: بازیکن
عامل فرعی: ندارد
شرایط اولیه: ندارد
روند اصلی: ۱. کاربر آدرس برنامه سرور بازی را می دهد. ۲. به سرور بازی متّصل شو. ۳. منتظر پیوستن بقیّه بازیکنان بمان.
شرایط نهایی: ندارد
روند جایگزین: ندارد

مورد کاربرد: GuessCharacter
شماره: ۴
توصیف اجمالی: پس از شروع بازی کاربر کاراکتری که را که حدس می زند در جلمه باشد را حدس می زند
عامل اصلی: بازیکن
عامل فرعی: ندارد
شرایط اولیه: همه برنامه های مشتری بازی به سرور متّصل شده‌اند و اطلاعات بازی دریافت شده.
روند اصلی: ۱. کاربر اطلاعاتی مبنی به امتیاز خود و بقیّه و کاراکترهای صحیح حدس زده شده را بر روی صفحه مشاهده می کند. ۲. شامل مورد کاربری SeeResults ۳. کاربر کاراکتر جدیدی را حدس می زند ۴. امتیاز جدید کاربر محاسبه می گردد. ۵. سرور از امتیازات جدید مطّلع می گردد. ۶. امتیازات جدید به کاربر نشان داده می شود. ۷. اگر بازی تمام شده ۷.۱. کاربر را مطّلع کن ۷.۲. سرور را از اتمام بازی مطّلع کن ۷.۳. از بازی خارج شو. ۸. منتظر کاراکتر وروردی کاربر شو.
شرایط نهایی: منتظر کاراکتر ورودی بعدی کاربر شو.
روند جایگزین: ندارد

مورد کاربرد: SeeResults
شماره: ۵
توصیف اجمالی: کاربر اطلاعاتی مبنی به امتیاز خود و بقیّه و کاراکترهای صحیح حدس زده شده را بر روی صفحه مشاهده می کند.
عامل اصلی: بازیکن
عامل فرعی: ندارد
شرایط اولیه: ندارد
روند اصلی: ۱. اطلاعات بازی را به روز کن ۲. امتیاز بازیکن حاری و دیگر بازیکنان را نشان بده.
شرایط نهایی: ندارد
روند جایگزین: ندارد

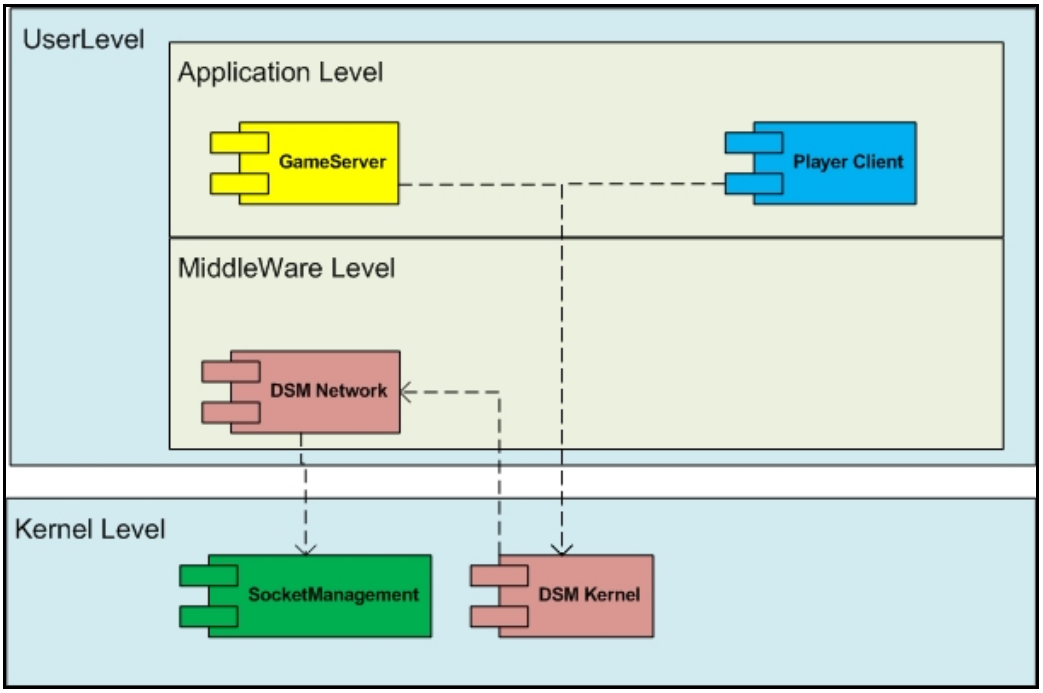
نمودار مؤلّفه

مقدمّه

در این مستند انواع مؤلّفه ها در سیستم نشان داده شده است. همچنین ارتباط این مؤلّفه ها و نحوه استفاده آنها از یکدیگر در جهت انجام مسئولیّت هایشان به نمایش گذاشته شده است. همچنین محصولات (artifacts) مرتبط با هر مؤلّفه و ارتباط خود این محصولات در نمودار دیگری نشان داده شده است.

نمودار مؤلّفه

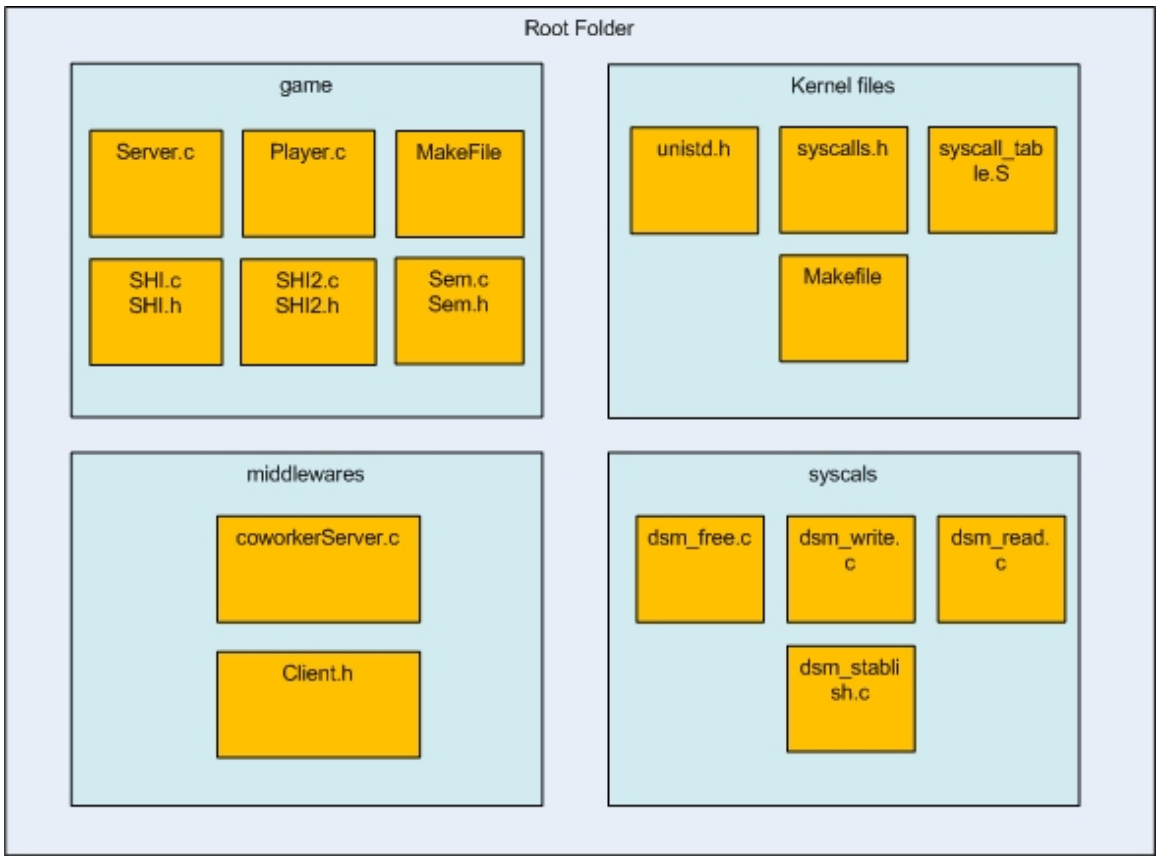
در این نمودار مؤلّفه ها و محصولات عمده که آنها را محقق می کنند نشان داده شده است. سیستم در قالب دو نمودار معرّفی شده است. اوّل نمودار مربوط به مدیر حافظه توزیع‌شده که به عنوان یک میان‌ابزار استفاده می شود نشان داده شده است. در نمودار دوّم تنها مؤلّفه های تعریف شده مربوط به خود بازی و نحوه استفاده آنها از میان‌ابزار ذکر شده نشان داده می‌شود.



شکل ۲ نمودار مؤلّفه سیستم

محصولات

در این نمودار محصولات (artifacts) ی که شامل محصولات دیگری است نشان داده شده است. این نمودار به عنوان نمودار تکمیلی جهت نمودار بالا قابل استفاده است. همچنین در نمودار انتقال به آن ارجاع داده می شود.


Root Folder
game
Server.c
Player.c
MakeFile
SHI.c
SHI.h
SHI2.c
SHI2.h
Sem.c
Sem.h
Kernel files
unistd.h
syscalls.h
syscall_table.S
Makefile
middlewares
coworkerServer.c
Client.h
syscals
dsm_free.c
dsm_write.c
dsm_read.c
dsm_stablish.c

نمودار فعالیت

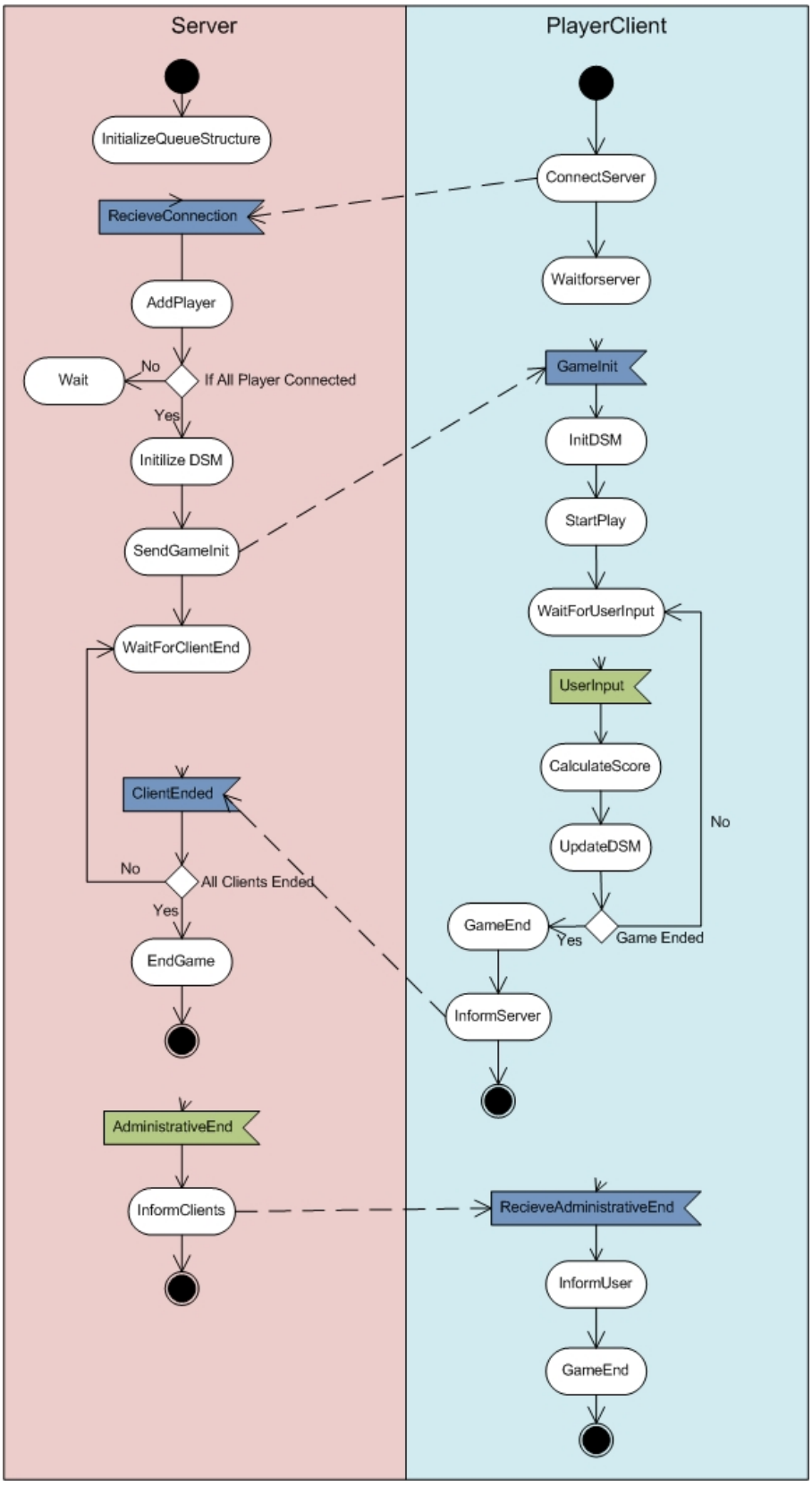
مقدمه

در این مستند نمودار فعّالیّت (Activity Diagram) و در واقع جریان کلّی اجرا شدن این برنامه آورده شده است. همچنین در بخش بعدی به تفصیل نحوه انجام هر یک از گره های پردازشی این نمودار آورده شده است.

نمودارهای فعّالیّت

جریان کلّی اجرای برنامه از دید کلان به صورت زیر نشان داده شده است.

جریان بازی


Server
PlayerClient
InitializeQueueStructure
ConnectServer
RecieveConnection
Waitforserver
AddPlayer
GameInit
Wait
No
If All Player Connected
Yes
InitDSM
Initilize DSM
StartPlay
SendGameInit
WaitForUserInput
WaitForClientEnd
UserInput
CalculateScore
ClientEnded
No
UpdateDSM
No
All Clients Ended
GameEnd
Yes
Game Ended
Yes
EndGame
InformServer
AdministrativeEnd
InformClients
RecieveAdministrativeEnd
InformUser
GameEnd

پیاده سازی نمودار فعالیت

مقدمّه

در این مستند فایل‌های اصلی لازم جهت تحقّق وظیفه موارد کاربرد مورد نظر در مستند موارد کاربرد پیوست طرّاحی شده‌اند. در واقع این بخش نحوه محقق سازی (پیاده‌سازی) نمودار فعالیّت مذکور در بالا را به وسیله فایلهایی که در بخش نمودار مؤلّفه توضیح نشان داده شده اند مشخّص می کند. همچنین بدیهی است که تأکید بیشتر در این بخش برروی نحوه پیاده سازی الگوریتم خود بازی می باشد.

فایل player.c و server.c

در ابتدای این فایل کدهایی برای مقداردهی اوّلیّه و ساختن حافظه به اشتراکگذاری شده محلّی در اجرای برنامه نوشته شده است

```
int main(int argc, char** argv) {
    //initialization
    int j=0;
    int maxP=1;
    pid_t child_pids[maxP];
    struct timespec delay;
    delay.tv_sec = maxsec - 1;
    delay.tv_nsec = 999999999;

    //init local shared memory
    init_shared(PAGE_MAX_SIZE,2);
    init_integers(1);
    //init semaphors
    init_semaphores(1);
    sem_init(serverWait,0);
```

در این برنامه دو Thread در نظر گرفته شده است به این شکل که یکی مسئول خواندن و به روزرسانی حافظه به اشتراک گذاشته شده توزیع شده است و دیگر مسئول مدیریت بازی و تعامل با کاربر. در ادامه قسمت اوّل نمودار فعّالیّت یعنی قسمت اتّصال به سرور و ثبت‌نام در بازی را مشاهده می کنید. این تابع یک اتّصال از نوع Datagram با برنامه سرور برقرار می کند. و حضور خود را در بازی اعلام می‌کند.

```
child_pids[j] = fork();
if (child_pids[j] == 0){
    //do the algorithm for entering the server
    sendMessageToGameServer("CNNT");
```

```
void sendMessageToGameServer(char* message)
{
    // Defining listen socket and connecting to it
    int receive_sd;
    struct sockaddr_in receive_addr;
    // The received message should contain pageID, coworker IP address,
    // and a pointer to beginning of page section on coworker address space
    char receive_msg[12];
    bzero(&receive_addr, sizeof(receive_addr));
    receive_addr.sin_family = AF_INET;
    receive_addr.sin_port = htons(LISTEN_PORT_ALLOC);
    receive_addr.sin_addr.s_addr = INADDR_ANY;
    receive_sd = socket(PF_INET, SOCK_DGRAM, 0);
    if(bind(receive_sd, &receive_addr, sizeof(receive_addr)) != 0){
        printf("Cannot bind to listen port\n");
        return -1;
    }

    // Defining send socket

    int sd;
    struct sockaddr_in addr;
    struct hostent *server_name;
    char buffer[1024];
    int len;

    memset(&addr, 0, sizeof(addr));

    addr.sin_family = AF_INET;
    addr.sin_port = htons(DATAGRAM_CONNECT_PORT);
    addr.sin_addr.s_addr = inet_addr(DSM_SERVER_IP_ADDRESS);

    sd = socket(AF_INET, SOCK_DGRAM, 0);
    //computing the size of the message to be sent to coworker
    int messageSize = 0;
    for(int j = 0; j < 32; j++)
    {
        if(message[j] == 0)
        {
            messageSize = j;
            break;
        }
    }

    //sending the connect message to server
    if(sendto(sd, message, messageSize + 1, 0, (struct sockaddr *) &addr, sizeof(addr)) < 0){
        printf("cannot connect to coworker%d\n", i);
        return -1;
    }
    close(sd);
```

اکنون به سراغ فایل server.c می رویم. در این فایل ابتدا عملیّات مقداردهی اوّلیّه به صورت زیر انجام می شود. همچنین در این بخش کنترل مقادیر ورودی کاربر نیز صورت می گیرد.

```
int main(int argc, char** argv) {
    if (argc == 3){
        //initialization
        int pnum=atoi(argv[1]);
        char* sentence=argv[2];
        int clients=0;
        //array that preservs clients address
        sockaddr_in clientsSockets [pnum] ;
        int clientsSocketsLen [pnum];
```

سرور طبق نمودار فعّالیّت منتظر همه بازیکن ها می شود. این کار با استفاده از ابزار socket به صورت زیر انجام شده است.

```
//busy while until all clients connect
while(clients<pnum) {
    //create sockets
    // Defining listen socket and connecting to it
    int receive_sd;
    struct sockaddr_in receive_addr;

    char receive_msg[12];
    bzero(&receive_addr, sizeof(receive_addr));
    receive_addr.sin_family = AF_INET;
    receive_addr.sin_port = htons(LISTEN_PORT_ALLOC);
    receive_addr.sin_addr.s_addr = INADDR_ANY;
    receive_sd = socket(PF_INET, SOCK_DGRAM, 0);


    //wait for client connect message
    for(ever)
    {
        if(bind(receive_sd, &receive_addr, sizeof(receive_addr)) != 0){
            printf("Cannot bind to listen port\n");
            return -1;
        }
        struct sockaddr_in sin;
        int sin_len = sizeof(sin);
        int len = recvfrom(receive_sd, msg, 32, 0, (struct sockaddr *)&sin, &sin_len);
        msg[len] = 0;
        if (executeMessage(msg, len)>0){
            //A client connected add it to clients;
            clientsSockets[clients]=sin;
            clientsSocketsLen[clients]=sin_len;
            clients++;
            receive_sd.close();
            break;
        }
    }
}
```

توابع استفاده شده در کد بالا به صورت زیر هستند:

```
int executeMessage(const char * msg, int len)
{
    //The message is either ALLOC, FREE, REPLICATE, EXPAND, READ, or COPY
    //The first 4 characters of DSM server message
    char 4firstMsgChars[4];
    for(int i = 0; i < 4; i++)
        4firstMsgChars[i] = msg[i];
    char seps[] = " ";

    if(strcmp(4firstMsgChars, "CNNT") == 0)
    {
        return 1;
    }
    if(strcmp(4firstMsgChars, "ENDD") == 0)
    {
        return 2;
    }
    return 0;
}
```

```
int getSizeOf(char * s){
    int messageSize = 0;
    for(int k = 0; k < 32; k++)
    {
        if(s[k] == 0)
        {
            messageSize = k-1;
            break;
        }
    }
    return messageSize;
}
```

بعد از آنکه همه بازیکن ها به بازی پیوستند، طبق نمودار سرور حافظه به اشتراک گذاشته شده توزیع شده را مقداردهی اوّلیّه می کند.

```
    //create initial values
    char sep = '#';
    char data[1024];
    int i
    for (i = 0;i<sizeof(sentence);i++){
        data[i]=sentence[i];
    }
    data[i++]=sep;
    char tempS [3];
    sprintf(tempS,"%d%c",pnum,sep);
    int tempSize = getSizeof(tempS);
    for (int k=0;k<tempSize;k++){
        data[i++]=tempS[k];
    }
    for (int j = 0;j<pnum;j++){
        char clientString [32];
        sprintf(clientString,"%d%c%d%c",j,sep,0,sep);
        int messageSize = getSizeof(clientString);
        for (int k=0;k<messageSize;k++){
            data[i++]=clientString[k];
        }
    }
    data[i]=0;

    //create shared memory and put initial values
    int pageID = dsm_stablish(1024,data,i);
```

این عمل به صورت زیر انجام می شود. ساختار page به اشتراک گذاشته شده به صورت زیر است

```
sentence#numberofClients#client1_ID#client1_Score#
client2_ID#client2_Score#. . .#
```

بعد از این باید به تمام بازیکن ها شروع بازی را اعلام کنیم

```
    //send start to all
    for (int numStartSent=0;numStartSent<pnum;numStartSent++){
        sendStart(pnum, clientsSockets,clientsSocketsLen );
    }
```

تابع استفاده شده در بالا به صورت زیر پیاده سازی شده است:

```
/*
    method that sends a START signal too all connected clients
*/
void sendStart(int clientsNum, sockaddr_in* clientsAddress, int* clientsAddressLen ){
    for (int i = 0; i<clientsNum; i++){
        // Defining send socket

        int sd;
        struct hostent *server_name;
        char buffer[1024];
        int len;

        sd = socket(AF_INET, SOCK_DGRAM, 0);

        //building the message to be sent to clients
        char message[32];
        sprintf(message, "STRT %d",i);
        //computing the size of the message to be sent to clients
        int messageSize = 0;
        for(int j = 0; j < 32; j++)
        {
            if(message[j] == 0)
            {
                messageSize = j;
                break;
            }
        }

        //sending the connect message to server
        if(sendto(sd, message, messageSize + 1, 0, (struct sockaddr *) &clientsAddress[i], clientsAddressLen[i]) < 0){
            printf("cannot connect to coworker%d\n", i);
            return -1;
        }
        close(sd);
    }
}
```

بدین وسیله به تمامی بازیکن هایی که درخواست پیوستن به بازی را داشته اند یک شماره یکتا داده می شود. و همراه پیغام شروع به آنها پیغام ارسال می گردد.

و حالا باز به طرف برنامه بازیکن می رویم. در ادامه تابع SendMessagetoGameServer داریم:

```
        //wait for server start signal
        for(ever)
        {
            struct sockaddr_in sin;
            int sin_len = sizeof(sin);
            int len = recvfrom(receive_sd, msg, 32, 0, (struct sockaddr *)&sin, &sin_len);
            msg[len] = 0;
            if (executeMessage(msg, len)>0){
                break;
            }
            //printf("The message is %s\n", msg);
        }

    }
```

همچنین تابع executeMessage که وظیفه تفسیر پیغام سرور را دارد به صورت زیر پیاده سازی شده است. این تابع ID داده شده توسّط سرور را از پیغام جدا می کند تا بعداً بتوان به وسیله آن امتیاز بازیکن را به روز کرد.

```
int executeMessage(const char * msg, int len)
{
    //The first 4 characters of Game server message
    char 4firstMsgChars[4];
    for(int i = 0; i < 4; i++)
        4firstMsgChars[i] = msg[i];
    char seps[] = " ";

    if(strcmp(4firstMsgChars, "STRT") == 0)
    {
        char sep[] = " ";
        char *token;
        token = strtok( msg, seps );
        token = strtok(NULL, seps);
        int myId = atoi(token);
        shi2_set(MY_ID_SHARED_INDEX,myId);
        return 1;
    }
    return 0;
}
```

بعد از این مرحله بازی رسماً شروع شده است. ولی قبل از آن لازم است تا مقدارهای اوّلیّه بازی را از حافظه به اشتراک گذاشته شده توزیع شده بخوانیم و داده های لازم را از آن اسخراج کنیم

```
//2: get data
char* data;
dsm_read(pageID,DSM_SERVER_IP_ADDRESS,data);
//parse data
char sentence[PAGE_MAX_SIZE];
char clientsScoreString[1000];
parseData(data,sentence,clientsScoreString);
shi_set(sentence,PAGE_MAX_SIZE,0);
shi_set(data,PAGE_MAX_SIZE,1);
//update screen using shared variable in shared memory
printf(clientsScoreString);
//signal semaphor serverWait
sem_signal(serverWait);  //
```

بعد از این قسمت، Thread جاری برنامه در یک حلقه مرتباً حافظه مشترک را چک می کند. در کارهای آینده جهت بهبود می توان invalidation را هم به این برنامه اضافه کرد

```
while (true){
    //sleep
    nanosleep(&delay,0);
    //read data
    dsm_read(pageID,DSM_SERVER_IP_ADDRESS,data);
    parseData(data,sentence,clientsScoreString);
    //update screen using shared variable in shared memory
    printf(clientsScoreString);
}
```

وحالا بخش مربوط به Thread بازی که در ابتدا به وسیله یک سمافور محلّی منتظر Thread ذکر شده در بالا می ماند و پس از آن به صورت زیر ادامه کار می دهد

```
//show waiting for server message and wait
printf("waitng for server");
//wait for server
sem_wait(serverWait);
char * sentence;
shi_get(sentence,PAGE_MAX_SIZE);
char * data;
shi_get(data,PAGE_MAX_SIZE);
//copy sentence
char sentence2 [PAGE_MAX_SIZE];
for (int i = 0;i<sizeof(sentence);i++){
    sentence2[i]=sentence[i];
}
int myId = shi2_get(MY_ID_SHARED_INDEX);
int score =0;
do{
    //1: wait for user input
    int user_char;
    scanf ("%c", &user_char);
    //calculate new grade using (answer)
    score+=calculateGrade(sentence2,user_char);
    //send grades to server using shared memory
    //update shared memory
    updateMyScore(score,data,myId);
    dsm_write(pageID,data, PAGE_MAX_SIZE, );
    //check if game is over?
    //eles go to 1
while (score == sizeof(sentence));
```

این عمل آنقدر ادامه می یابد تا بازیکن تمامی حروف را حدس بزند. تابع زیر مسئول به روز رسانی امتیاز بازیکن است:

```
/*
    finds and updates my score in shared page.
*/
void updateMyScore(int score,char * data,int myId){
    char dataCopy [ sizeof(data)];
    for (int i=0;i<sizeof(data);i++){
        dataCopy[i] = data [i];
    }
    char sep[] = "#";
    char *token;
    int bufferPos=0;
    token = strtok( msg, seps );
    //skip sentence
    token = strtok(NULL, seps);
    bufferPos+=sizeof(token+1);
    int pnum =0;
    token = strtok(NULL, seps)
    bufferPos+=sizeof(token+1);;
    pnum = atoi(token);
    for (int i=0;i<pnum ; i++){
        int clientID=0;
        token = strtok(NULL, seps);
        clientID = atoi(token);
        if (clientID==myId){
            //my record in DSM found
            //update it
            char tempS[10];
            sprintf(tempS,"%d",score);
            int j;
            for(j=0;j<sizeof(tempS);j++){
                data[bufferPos+j]=tempS[j];
            }
            int newBufferPos =bufferPos+j;
            bufferPos+=sizeof(token+1);

            //copy tail of datacopy to data = shift
            for (int k = bufferPos; k<sizeof(dataCopy)-j; k++){
                data[k]=datacopy[newBufferPos++];
            }
            break;
        }
        //skip score field
        token = strtok(NULL, seps);
    }
}
```

پس از اتمام بازی همانطور که در نمودار فعالیت نیز آمده است بازیکن ها یک پیغام کنترلی پایانی به سرور می فرستند و از حافظه خارج می شوند

```
    //send end signal to server
    sendMessageToGameServer("ENDD");
    //finalizing
    clean_semaphores();
    clean_shared();
    return 0;
}
```

تابع SendMessageToGameServer قبلاً در بالا ذکر شده بود. در بخش بعدی نحوه پیاده سازی بخش پایانی در سمت سرور آورده شده است:

```
        //busy while until all clients end
        while(clients<pnum) {
            //create sockets
            // Defining listen socket and connecting to it
            int receive_sd;
            struct sockaddr_in receive_addr;
            // The received message should contain pageID, coworker IP address,
            // and a pointer to beginning of page section on coworker address space
            char receive_msg[12];
            bzero(&receive_addr, sizeof(receive_addr));
            receive_addr.sin_family = AF_INET;
            receive_addr.sin_port = htons(LISTEN_PORT_ALLOC);
            receive_addr.sin_addr.s_addr = INADDR_ANY;
            receive_sd = socket(PF_INET, SOCK_DGRAM, 0);


            //wait for client end signal
            for(ever)
            {
                if(bind(receive_sd, &receive_addr, sizeof(receive_addr)) != 0){
                    printf("Cannot bind to listen port\n");
                    return -1;
                }
                struct sockaddr_in sin;
                int sin_len = sizeof(sin);
                int len = recvfrom(receive_sd, msg, 32, 0, (struct sockaddr *)&sin, &sin_len);
                msg[len] = 0;
                if (executeMessage(msg, len)>0){
                    break;
                }
                //printf("The message is %s\n", msg);
            }
            clients++;


            receive_sd.close();
            //wait for all

        }

        //finalizing
        //delete shared memory structures
        dsm_free(pageID);
        return 0;
    }
    else{
      printf("not enough argument!!");
      return 1;
    }

}
```

پیاده سازی حافظه اشتراکی توزیع شده

ابتدا پوشه kernel files که مربوطه به فایلهای تنظیمی system call ها است توضیح داده می شود:

به فایل unistd.h کد های زیر اضافه شده یا تغییر داده می شوند:

```
#define __NR_epoll_pwait    319
#define __NR_dsm_alloc      320
#define __NR_dsm_free       321
#define __NR_dsm_read       322
#define __NR_dsm_write      323
#ifdef __KERNEL__

//#define NR_syscalls 320
#define NR_syscalls 324
```

سپس خطوط زیر به فایل syscall.h به شکل زیر اضافه می شوند:

```
asmlinkage long sys_dsm_stablish(int pageID, unsigned char * _pageData, int _pageSize);
asmlinkage long sys_dsm_free(int pageID);
asmlinkage long sys_dsm_read(int pageID, unsigned char * address, char* buffer);
asmlinkage long sys_dsm_write(int pageID, unsigned char * _pageData, int _pageSize, char* serverIP);
```

همچنین در فایل syscall_table.S نیز باید به صورت زیر نام system call های تعریف شده بدهیم:

```
.long sys_epoll_pwait
.long sys_dsm_stablish      /* 320 */
.long sys_dsm_free
.long sys_dsm_read
.long sys_dsm_write
```

در نهایت در Makefile مربوط به کرنل فایل های خود را نیز اضافه کنیم.

درکنار این system call ها برنامه ای به شکل یک میان ابزار هم وجود دارد که کار مدیریت page های حافظه و ارسال داده ها روی شبکه در طرف سرور را انجام می دهد. به این برنامه هم system call های روی سرور و هم روی ماشین های مشتری پیغام می دهند و کد آن به صورت زیر است:

```
int main(void)
{
    int sd;
    struct sockaddr_in addr;
    char msg[32];
    bzero(&addr, sizeof(addr));

    addr.sin_family = AF_INET;
    addr.sin_port = htons(LISTEN_PORT);
    addr.sin_addr.s_addr = INADDR_ANY;

    sd = socket(PF_INET, SOCK_DGRAM, 0);

    if(bind(sd, &addr, sizeof(addr)) != 0){
        printf("Cannot bind to listen on port 8989\n");
        return -1;
    }
    for(ever)
    {
        struct sockaddr_in sin;
        int sin_len = sizeof(sin);
        int len = recvfrom(sd, msg, 32, 0, (struct sockaddr *)&sin, &sin_len);
        msg[len] = 0;
        executeMessage(msg, len, sin);
        //printf("The message is %s\n", msg);
    }
    close(sd);
    return 0;
}
```

همچنین تابع executeMessage که در واقع کار اصلی را انجام می دهد به صورت زیر پیاده سازی شده است:

```
int executeMessage(const char * msg, int len, sockaddr_in sin)
{
    //The message is either ALLOC, FREE, READ, or WRITE
    //The first 4 characters of DSM server message
    char 4firstMsgChars[4];
    for(int i = 0; i < 4; i++)
        4firstMsgChars[i] = msg[i];
    char seps[] = " ";

    if(strcmp(4firstMsgChars, "ALLO") == 0)
    {
        char *token;
        int pID;
        int pSize;
        token = strtok( msg, seps );
        token = strtok(NULL, seps);
        pID = atoi(token);
        token = strtok(NULL, seps);
        pSize = atoi(token);
        allocatePage(pID, pSize);
    }
    else if(strcmp(4firstMsgChars, "FREE") == 0)
    {
        //The command was FREE
        char *token;
        int pID;
        token = strtok( msg, seps );
        token = strtok(NULL, seps);
        pID = atoi(token);
        freePage(pID);
    }

    else if(strcmp(4firstMsgChars, "READ") == 0)
    {
        //The command was READ
        char *token;
        int pID;
        token = strtok( msg, seps );
        token = strtok(NULL, seps);
        pID = atoi(token);
        char * buffer
        readPage(pID,buffer);

        // Defining send socket
        int sd;
        struct hostent *server_name;
        char buffer[1024];
        int len;

        memset(&sin, 0, sizeof(sin));

        sd = socket(AF_INET, SOCK_DGRAM, 0);

        //computing the size of the message to be sent to client
        int messageSize = 0;
        for(int j = 0; j < 32; j++)
        {
            if(buffer[j] == 0)
            {
                messageSize = j;
                break;
            }
        }
        //sending the page_write message to coworker
        if(sendto(sd, buffer, messageSize + 1, 0, (struct sockaddr *) &sin, sizeof(sin)) < 0){
            printf("cannot connect to client%d\n", i);
            return -1;
        }
```

```
        close(sd);
    }else if(strcmp(4firstMsgChars, "WRIT") == 0)
    {
        //The command was READ
        char *token;
        int pID;
        int bufferPos=0;
        token = strtok( msg, seps );
        token = strtok(NULL, seps);
        bufferPos = 6+sizeof(token);
        pID = atoi(token);
        writePage(pID,bufferPos,msg,len);
    }
    else
        return -1;
}
```

پیاده سازی توابع استفاده شده در تابع مذکور نیز به صورت زیر است:

```
void allocatePage(int _pageID, int _pageSize)
{
    unsigned char * _pageData = malloc(_pageSize);
    PageInfo * info = getPageInfo(_pageID);
    //info->pageID = _pageID;
    info->pageSize = _pageSize;
    info->pageData = _pageData;
    info->isReleased = false;
}

void freePage(int _pageID)
{
    PageInfo * info = getPageInfo(_pageID);
    free(info->pageData);
    info->isReleased = true;
}

int readPage(int _pageID, char* buffer)
{
    PageInfo * pageInfo = getPageInfo(_pageID);
    buffer = malloc(pageInfo->pageSize);
    memcpy(replicatedPageInfo->pageData, pageInfo->pageData, pageInfo->pageSize);
    return 1;
}

void writePage(int pid, int bufferPos, char* msg, int len){
    PageInfo * pageInfo = getPageInfo(_pageID);
    if(pageInfo->pageData != 0)
        free(pageInfo->pageData);
    char newPage[len];
    for (int i=0;i<len-bufferPos;i++){
        newPage[i]=msg[bufferPos+i];
    }
    pageInfo->pageData = newPage;
    pageInfo->pageSize = len - bufferPos;
    pageInfo->isReleased = false;
}
```

اکنون خود system call ها توضیح داده می شوند:

:dsm_stablish

این کد به میان ابزار دستور می دهد که یک page را در نظر بگیرد و سپس شماره page تولید شده را جهت رهگیری های بعدی برمی گرداند.

```
asmlinkage int sys_dsm_stablish(int pageID, unsigned char * _pageData, int _pageSize)
{
    printk(KERN_EMERG "dsm_stablish is running!");
    // Defining send socket
    int currentPageID = getNewPageID();
    int memSectionSize = _size;
    int sd;
    struct sockaddr_in addr;
    struct hostent *server_name;
    char buffer[1024];
    int len;

    memset(&addr, 0, sizeof(addr));

    addr.sin_family = AF_INET;
    addr.sin_port = htons(DATAGRAM_CONNECT_PORT);
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    sd = socket(AF_INET, SOCK_DGRAM, 0);

    //building the message to be sent to coworker
    char message[32];
    sprintf(message, "ALLOC %d %d", currentPageID, memSectionSize);
    //computing the size of the message to be sent to coworker
    int messageSize = 0;
    for(int j = 0; j < 32; j++)
    {
        if(message[j] == 0)
        {
            messageSize = j;
            break;
        }
    }

    //sending the page_alloc message to coworker
    if(sendto(sd, message, messageSize + 1, 0, (struct sockaddr *) &addr, sizeof(addr)) < 0){
        printf("cannot connect to coworker%d\n", i);
        return -1;
    }

    //building the message to be sent to coworker
    char message2[1200];
    sprintf(message2, "WRIT %d", currentPageID);
    //computing the size of the message to be sent to coworker
    messageSize = 0;
    for(int j = 0; j < 32; j++)
    {
        if(message[j] == 0)
        {
            messageSize = j;
            break;
        }
    }
    //copy page data to message
    for (int j =0; j < _pageSize; j++){
        message2[++messageSize]=_pageData[j];
    }
    //sending the page_write message to coworker
    if(sendto(sd, message2, messageSize + 1, 0, (struct sockaddr *) &addr, sizeof(addr)) < 0){
        printf("cannot connect to coworker%d\n", i);
        return -1;
    }
    close(sd);

    return currentPageID;
```

:dsm_read

این کد که در طرف مشتری‌ها صدا زده می‌شود جهت خواندن یک page به اشتراک گذاری شده در روی سرور مرکزی است. پس از اتّصال به میان ابزار اجرا شده روی سرور و گرفتن اطلاعات آنها را در بافری که از طرف کاربر پاس شده کپی می کند.

```
asmlinkage int sys_dsm_read(int pageID, unsigned char * address, char* buffer)
{
    printk(KERN_EMERG "dsm_read is running!");

    /create socket to middleware
    // Defining listen socket and connecting to it
    int receive_sd;
    struct sockaddr_in receive_addr;
    // The received message should contain pageID, coworker IP address,
    // and a pointer to beginning of page section on coworker address space
    char receive_msg[12];
    bzero(&receive_addr, sizeof(receive_addr));
    receive_addr.sin_family = AF_INET;
    receive_addr.sin_port = htons(LISTEN_PORT_ALLOC);
    receive_addr.sin_addr.s_addr = INADDR_ANY;
    receive_sd = socket(PF_INET, SOCK_DGRAM, 0);
    if(bind(receive_sd, &receive_addr, sizeof(receive_addr)) != 0){
        printf("Cannot bind to listen port\n");
        return -1;
    }

    // Defining send socket
    int sd;
    struct sockaddr_in addr;
    struct hostent *server_name;
    char buffer[1024];
    int len;

    memset(&addr, 0, sizeof(addr));

    addr.sin_family = AF_INET;
    addr.sin_port = htons(DATAGRAM_CONNECT_PORT);
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    sd = socket(AF_INET, SOCK_DGRAM, 0);
```

```
//building the message to be sent to coworker
char message2[32];
sprintf(message2, "READ %d", pageID);
//computing the size of the message to be sent to coworker
int messageSize = 0;
for(int j = 0; j < 32; j++)
{
    if(message[j] == 0)
    {
        messageSize = j;
        break;
    }
}
//copy page data to message
for (int j =0; j < _pageSize; j++){
    message2[++messageSize]=_pageData[j];
}
//sending the page_write message to coworker
if(sendto(sd, message2, messageSize + 1, 0, (struct sockaddr *) &addr, sizeof(addr)) < 0){
    printf("cannot connect to coworker%d\n", i);
    return -1;
}
close(sd);

//wait for coworker datas
for(ever)
{
    struct sockaddr_in sin;
    int sin_len = sizeof(sin);
    int len = recvfrom(receive_sd, msg, 32, 0, (struct sockaddr *)&sin, &sin_len);
    msg[len] = 0;
    if (len>0){
        buffer = msg;
        break;
    }
}

return 0;
```

:dsm_write

این کد برای پیاده سازی عمل نوشتن توسّط مشتری ها و در واقع بروزرسانی حافظه مشترک است. این کد با اتّصال به میان‌ابزار حافظه مشترک را به روز می کند.

```
asmlinkage int sys_dsm_write(int pageID, unsigned char * _pageData, int _pageSize, char* serverIP)
{
    printk(KERN_EMERG "dsm_write is running!");

    // Defining send socket
    int sd;
    struct sockaddr_in addr;
    struct hostent *server_name;
    char buffer[1024];
    int len;

    memset(&addr, 0, sizeof(addr));

    addr.sin_family = AF_INET;
    addr.sin_port = htons(DATAGRAM_CONNECT_PORT);
    addr.sin_addr.s_addr = inet_addr(serverIP);

    sd = socket(AF_INET, SOCK_DGRAM, 0);

    //building the message to be sent to coworker
    char message2[1200];
    sprintf(message2, "WRIT %d", currentPageID);
    //computing the size of the message to be sent to coworker
    int messageSize = 0;
    for(int j = 0; j < 32; j++)
    {
        if(message[j] == 0)
        {
            messageSize = j;
            break;
        }
    }
    //copy page data to message
    for (int j =0; j < _pageSize; j++){
        message2[++messageSize]=_pageData[j];
    }

    //sending the page_write message to coworker
    if(sendto(sd, message2, messageSize + 1, 0, (struct sockaddr *) &addr, sizeof(addr)) < 0){
        printf("cannot connect to coworker%d\n", i);
        return -1;
    }
    close(sd);

    return 0;
}
```

:dsm_free

این قسمت از برنامه وظیفه ارسال دستور آزاد سازی حافظه گرفته شده به میان‌ابزار (یا مدیر حافظه به اشتراک گذاشته شده) را انجام می دهد. کد آن در ادامه آورده شده است:

```
// the function is to free a page.
asmlinkage int sys_dsm_free(int pageID)
{
    printk(KERN_EMERG "dsm_free is running!");


    // Defining send socket
    int sd;
    struct sockaddr_in addr;
    struct hostent *server_name;
    char buffer[1024];
    int len;

    memset(&addr, 0, sizeof(addr));

    addr.sin_family = AF_INET;
    addr.sin_port = htons(DATAGRAM_CONNECT_PORT);
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    sd = socket(AF_INET, SOCK_DGRAM, 0);

    //building the message to be sent to coworker
    char message2[32];
    sprintf(message2, "FREE %d", currentPageID);
    //computing the size of the message to be sent to coworker
    int messageSize = 0;
    for(int j = 0; j < 32; j++)
    {
        if(message[j] == 0)
        {
            messageSize = j;
            break;
        }
    }

    //copy page data to message
    for (int j =0; j < _pageSize; j++){
        message2[++messageSize]=_pageData[j];
    }
    //sending the page_write message to coworker
    if(sendto(sd, message2, messageSize + 1, 0, (struct sockaddr *) &addr, sizeof(addr)) < 0){
        printf("cannot connect to coworker%d\n", i);
        return -1;
    }
    close(sd);

    return 0;
}
```

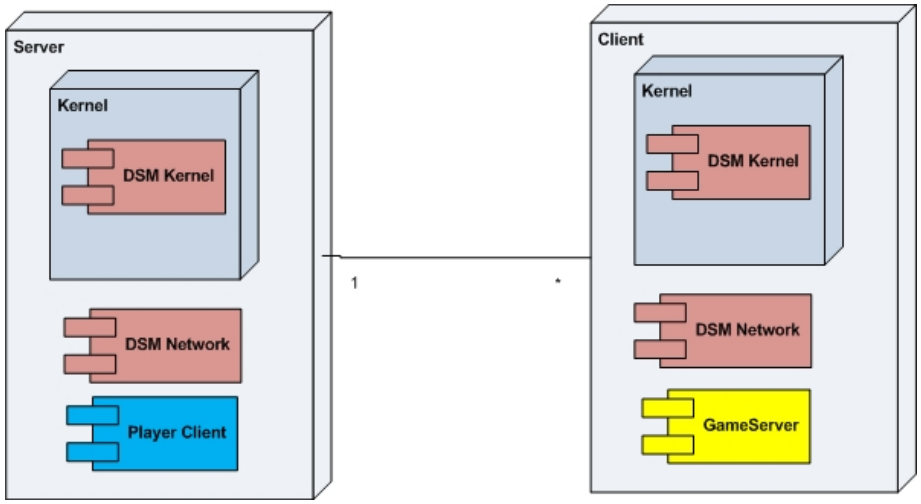
نمودار استقرار

مقدّمه

در این مستند انواع گره‌های محاسباتی دخیل در این سیستم و محصولات (artifacts) که بر روی آنها قرار می گیرد نشان داده شده است. همچنین ارتباط میان این گره‌های به شکل یک پیوند برقرار شده است.

نمودار استقرار

همانطور که در نمودار نشان داده شده دو نوع گره پردازش داریم. اوّل گره سرور که تنها یکی از آن در کلّ سیستم وجود دارد و برنامه سرور بازی روی آن اجرا می گردد. نوع گره دوّم که به تعداد بازیکن ها وجود خواهد داشت اجرا کننده بخش اصلی مربوط به بازی در سیستم می‌باشد. همچنین در این پروژه مؤلّفه ای به نام مدیر حافظه توزیع شده وجود دارد که در تمام سیستم ها وجود دارد و وظیفه مدیریت حافظه توزیع شده را انجام می دهد و به عنوان یک میان‌ابزار عمل می‌کند.



شکل ۳ : نمودار انتقال سیستم دستگاه خودپرداز

لغت نامه پروژه

مقدمه

در این مستند مشخصات سعی شده است کلیّه واژه‌های مورد استفاده در مستندات پروژه نرم‌افزار سیستم خودپرداز بانک با فرمت استاندارد آورده شود.

ب

نام	بازیکن
نام مستعار	Player

نام	برنامه سرور بازی
نام مستعار	Server
توصیف محتویات/عملکرد	پردازه اجرا شده به عنوان سرور بازی

نام	برنامه سرور مشتری
نام مستعار	Client
توصیف محتویات/عملکرد	پردازه اجرا شده به عنوان مشتری بازی

ت

نام	توصیف اجمالی
نام مستعار	Brief Description

ر

نام	روند اصلی
نام مستعار	Main Flow
محل استفاده	

نام	روند جایگزین
نام مستعار	Alternative Flow
محل استفاده	

ز

نام	زیرسیستم
نام مستعار	SubSystem
محل استفاده	

ش

نام	شامل
نام مستعار	Include

نام	شرایط اولیه
نام مستعار	Precondition

نام	شرایط نهایی
نام مستعار	Postcondition

نام	شماره
نام مستعار	ID

نام	شماره والد
نام مستعار	Parent ID

ع

نام	عامل
نام مستعار	Actor، عملگر، بازیگر

نام	عامل اصلی
نام مستعار	Primary Actor
محل استفاده	
توصیف محتویات/عملکرد	

نام	عامل فرعی
نام مستعار	Secondary Actor

ک

نام	کرنل
نام مستعار	Kernel، هسته لینوکس

م

نام	مدیر سیستم
نام مستعار	Admin

نام	مدیر حافظه توزیع شده
نام مستعار	Distributed Shared Memory
محل استفاده	
توصیف محتویات/عملکرد	سیستم هماهنگ کننده حافظه توزیع شده، حافظه اشتراکی توزیع شده، حافظه به اشتراک گذاشته شده توزیع شده

نام	میان‌ابزار
نام مستعار	Middleware

ی

نام	یوزکیس
نام مستعار	Usecase، موردکاربرد
محل استفاده	