**OS Term Project 1: Scheduling Simulation**

고려대학교 컴퓨터학과

2015410113 신채호

1. **구현한 기능들**
2. fork()를 통한 10개의 자식 프로세스 생성
3. CPU burst time, I/O burst time 랜덤 설정
4. setitimer()를 통한 0.5초 주기의 Timer 설정
5. msgq를 통한 부모, 자식 프로세스들간의 IPC
6. Double linked list 방식의 Queue 생성
7. 1초 quantum의 Round-Robin을 통한 CPU Scheduling
8. CPU Scheduling이 끝난 Process들은 똑같이 Round-Robin을 통한 I/O Scheduling
9. sprintf, open, write를 통한 Schedule\_dump.txt 생성
10. **코드 설명**

개발 환경: Oracle VM VirtualBox – Ubuntu (64-bit)

사용언어: C 언어

실행의 편리성을 위해 Time Quantum은 1초이지만 1번의 Quantum마다 2의 CPU\_burst\_time 및 I/O\_burst\_time이 줄도록 설정.

1. 스케줄링 전 준비 – open() 함수를 통해 Schedule\_dumpt.txt를 열고 텍스트를 쓸 준비를 함. IPC를 위해 자식들이 각자 message queue를 쓸 수 있도록 10개의 message queue를 msgget 하고 부모에서 쓸 msgbuf를 초기화 시킴. 타이머를 위해 500000us(0.5초)마다 timer\_handler() 함수를 호출하는 타이머 설정(setitimer()). Timer\_handler()는 tick 마다 tick\_num과 time\_count를 1씩 증가 시키는데 tick\_num은 총 시간을 위한 변수고, time\_count는 1초마다 스케줄링 위한 변수. 스케줄링에 쓸 readyQueue, waitQueue init 후 readyQueue 에는 자식 프로세스들 CPU\_burst\_time과 몇 번째 자식인지 아는 Node를 10개 추가.

*int fd;*

*int written;*

*fd = open("/home/shin/Result.txt", O\_RDWR);*

*if(fd==-1)*

*{*

*printf("openError");*

*return 1;*

*}*

*// set msgq*

*key\_t key\_id[10];*

*struct msgbuf sndbuf, rcvbuf;*

*memset(sndbuf.mtext, 0x00, sizeof(sndbuf.mtext));*

*memset(rcvbuf.mtext, 0x00, sizeof(rcvbuf.mtext));*

*sndbuf.msgtype=2;*

*sndbuf.data =0;*

*rcvbuf.msgtype=2;*

*rcvbuf.data=0;*

*int k=0;*

*int i=0;*

*for(k=0;k<10;k++)*

*{*

*key\_id[k]= msgget((key\_t)(1234+k), IPC\_CREAT|0666);*

*if(key\_id[k] ==-1)*

*{*

*perror("msgget error");*

*exit(0);*

*}*

*}*

*//set timer*

*struct sigaction sa;*

*struct itimerval timer;*

*time\_count =0;*

*tick\_num =0;*

*memset(&sa, 0, sizeof(sa));*

*sa.sa\_handler = &timer\_handler;*

*sigaction(SIGVTALRM, &sa, NULL);*

*timer.it\_value.tv\_sec =0;*

*timer.it\_value.tv\_usec = 100000\* TIMEQUANTUM;*

*timer.it\_interval.tv\_sec = 0;*

*timer.it\_interval.tv\_usec = 100000 \* TIMEQUANTUM;*

*setitimer(ITIMER\_VIRTUAL, &timer, NULL);*

ProcessNum = 10;

int runningProcess = ProcessNum;

*//init readyqueue & waitqueue*

*queue readyQueue;*

*queue waitQueue;*

*InitQueue(&readyQueue);*

*InitQueue(&waitQueue);*

*for(i=0;i<ProcessNum;i++)*

*{*

*Enqueue(&readyQueue, CreateNode(CPU\_burst\_time[i], i));*

*}*

1. 자식 프로세스의 총 개수를 가질 ProcessNum 변수 10으로 초기화, CPU\_burst\_time, IO\_burst\_time이 다 끝나지 않은 프로세스의 수를 가질 runningProcess 변수 10으로 초기화 후, 10개의 자식 프로세스들 fork().

*ProcessNum = 10;*

*int runningProcess = ProcessNum;*

*for(i=0;i<ProcessNum;i++)*

*pid[i] = fork();*

1. Fork() 후 부모프로세스는 스케줄링 시작. runningProcess가 0이 될 때까지 스케줄링. readyQueue에 노드가 존재하면, 큐에서 하나 뽑아서(Dequeue()) 그 노드가 몇 번째 자식인지 파악 후 그 자식의 msgq key\_id에 차례를 알리는 메시지 보내고(message\_send()), 부모가 카운트 하는 자식의 CPU\_burst\_time 2만큼 감소. 만약 큐에서 뽑은 노드의 CPU\_burst\_time이 0보다 작은 경우 자식 프로세스가 메시지 보내길 기다리다가, 메시지가 오면 I/O를 위해 waitQueue로 보냄(Enqueue). waitQueue도 readyQueue와 똑같은 방식이지만 자식 프로세스와 통신은 안하고 tick마다 다음 자식 프로세스의 I/O\_burst\_time 감소. 모든 자식 프로세스 끝나면 (runningProcess가 0이되면) parent도 끝냄.

*while(runningProcess>0)*

*{*

*if(time\_count == 2)*

*{*

*if((readyQueue.count)>0)*

*{*

*nod = Dequeue(&readyQueue);*

*sndbuf.data = (int)(nod->pid);*

*sndbuf.msgtype = (long)(sndbuf.data+2);*

*if(CPU\_burst\_time[sndbuf.data]>0)*

*{*

*//Send message to child & decrease CPU\_burst*

*message\_send(key\_id[sndbuf.data], &sndbuf);*

*CPU\_burst\_time[sndbuf.data]-=2;*

*printf("Parentcheck: %d, %d \n",sndbuf.data,CPU\_burst\_time[sndbuf.data]);*

*fflush(stdout);*

*time\_count =0;*

*Enqueue(&readyQueue, nod);*

*}*

*else*

*{*

*//If CPU\_burst is zero move to waitQueue and do I/O*

*CPU\_burst\_time[sndbuf.data]=0;*

*if(child\_msg[sndbuf.data]==0)*

*{*

*if(msgrcv(key\_id[sndbuf.data], &rcvbuf, sizeof(struct msgbuf), 0, 0)==-1)*

*{*

*perror("msgrcv error");*

*exit(0);*

*}*

*child\_msg[sndbuf.data]=1;*

*Enqueue(&waitQueue, nod);*

*printf("in IO");*

*fflush(stdout);*

*}*

*}*

*if(CPU\_burst\_time[sndbuf.data]<1) CPU\_burst\_time[sndbuf.data]=0;*

*sprintf(buf, "Time: %d, Child No.%d gets CPU time, Remaining:%d, Run Queue: ", tick\_num/2, sndbuf.data, CPU\_burst\_time[sndbuf.data]);*

*written = write(fd, buf, strlen(buf));*

*memset(buf, 0x00, 256);*

*for(i=0;i<readyQueue.count;i++)*

*{*

*nod = Dequeue(&readyQueue);*

*sprintf(buf, "%d ", nod->pid);*

*written = write(fd, buf, strlen(buf));*

*Enqueue(&readyQueue, nod);*

*}*

*}*

*else*

*{*

*sprintf(buf, "Time: %d, ",tick\_num/2);*

*written = write(fd, buf, strlen(buf));*

*memset(buf, 0x00, 256);*

*}*

*sprintf(buf, "IO Wait Queue: ");*

*written = write(fd, buf, strlen(buf));*

*memset(buf, 0x00, 256);*

*if(waitQueue.count>0)*

*{*

*//Schedule I/O queue*

*nodT=Dequeue(&waitQueue);*

*IO\_burst\_time[nodT->pid]-=2;*

*printf("\nIO: %d \n", IO\_burst\_time[nodT->pid]);*

*fflush(stdout);*

*if(IO\_burst\_time[nodT->pid]<1)*

*{*

*IO\_burst\_time[nodT->pid]=0;*

*child\_msg[nodT->pid]=2;*

*runningProcess--;*

*}*

*else Enqueue(&waitQueue, nodT);*

*for(i=0;i<waitQueue.count;i++)*

*{*

*nodT = Dequeue(&waitQueue);*

*sprintf(buf, "%d ", nodT->pid);*

*written = write(fd, buf, strlen(buf));*

*Enqueue(&waitQueue, nodT);*

*}*

*}*

*sprintf(buf, "\n");*

*written = write(fd, buf, strlen(buf));*

*memset(buf, 0x00, 256);*

*printf("\n runningPro: %d \n", runningProcess);*

*fflush(stdout);*

*time\_count =0;*

*}*

*}*

*printf("finish");*

*fflush(stdout);*

*close(fd);*

*return 0;*

1. Fork() 후 자식 프로세스는 부모와의 통신을 위한 msgq를 msgget하고, CPU\_burst\_time 이 0이 되기 전까지 msgrcv를 통해 메시지 큐에 메시지가 오기를 기다리는 상태 반복(msgrcv 의 flag 0으로 설정). 부모로부터 메시지 받으면 자식이 카운트하고 있는 자신의 CPU\_burst\_time 2만큼 감소. 만약 자신의 CPU\_burst\_time이 0이 되면 부모에게 메시지 보내서 I/O 큐에 들어가고 자신은 특정 시간만큼 sleep()후 exit().

*else if(pid[i] == 0)*

*{*

*//Child Process i*

*printf("Child Process No.%d PID: %d ", i, (int)getpid() );*

*printf("CPUTIME:%d \n", CPU\_burst\_time[i]);*

*struct msgbuf sndbufT, rcvbufT;*

*int Num = i;*

*memset(sndbufT.mtext, 0x00, sizeof(sndbuf.mtext));*

*sndbufT.data = Num;*

*sndbufT.msgtype = i+2;*

*rcvbufT.data =0;*

*rcvbufT.msgtype = i+2;*

*key\_t key\_idt;*

*key\_idt = msgget((1234+i), IPC\_CREAT|0666);*

*if(key\_idt<0)*

*{*

*perror("Child msgget error");*

*exit(0);*

*}*

*while(CPU\_burst\_time[Num]>0)*

*{*

*if(msgrcv( key\_idt, &rcvbufT, sizeof(struct msgbuf), 0, 0) == -1)*

*{*

*perror("msgrcv error");*

*exit(0);*

*}*

*CPU\_burst\_time[Num]-=2;*

*printf("\n Num: %d, left: %d \n", Num, CPU\_burst\_time[Num]);*

*fflush(stdout);*

*}*

*sndbufT.data = IO\_burst\_time[i];*

*message\_send(key\_id[i], &sndbufT);*

*printf("child IO");*

*fflush(stdout);*

*sleep(IO\_burst\_time[Num]+30);*

*printf("exit");*

*fflush(stdout);*

*exit(100+i);*

*}*

1. **Schedule\_dump.txt 예시**

실제 파일은 Blackboard에 코드와 함께 제출.

처음에 Fork()시 자식 번호, CPU\_burst\_time, I/O\_burst time 출력.

예시:

Child No.0, CPU Time:63, IO Time:6

Child No.1, CPU Time:77, IO Time:5

Child No.2, CPU Time:73, IO Time:5

그 후 time tick(0.5초) 2번마다 1번씩 스케줄링 및 프린트. Time, 자식 번호, 스케줄링 한 프로세스의 남은 CPU time, run queue, wait queue.

처음 10개 예시:

Time: 1, Child No.0 gets CPU time, Remaining:61, Run Queue: 1 2 3 4 5 6 7 8 9 0 IO Wait Queue:

Time: 2, Child No.1 gets CPU time, Remaining:75, Run Queue: 2 3 4 5 6 7 8 9 0 1 IO Wait Queue:

Time: 3, Child No.2 gets CPU time, Remaining:71, Run Queue: 3 4 5 6 7 8 9 0 1 2 IO Wait Queue:

Time: 4, Child No.3 gets CPU time, Remaining:64, Run Queue: 4 5 6 7 8 9 0 1 2 3 IO Wait Queue:

Time: 5, Child No.4 gets CPU time, Remaining:67, Run Queue: 5 6 7 8 9 0 1 2 3 4 IO Wait Queue:

Time: 6, Child No.5 gets CPU time, Remaining:60, Run Queue: 6 7 8 9 0 1 2 3 4 5 IO Wait Queue:

Time: 7, Child No.6 gets CPU time, Remaining:68, Run Queue: 7 8 9 0 1 2 3 4 5 6 IO Wait Queue:

Time: 8, Child No.7 gets CPU time, Remaining:61, Run Queue: 8 9 0 1 2 3 4 5 6 7 IO Wait Queue:

Time: 9, Child No.8 gets CPU time, Remaining:58, Run Queue: 9 0 1 2 3 4 5 6 7 8 IO Wait Queue:

Time: 10, Child No.9 gets CPU time, Remaining:70, Run Queue: 0 1 2 3 4 5 6 7 8 9 IO Wait Queue:

위의 예시는 아직 CPU time이 하나도 끝나지 않아 Wait Queue에 아무것도 출력이 되지 않지만 CPU time이 끝난 프로세스가 있는 경우에는 Wait Queue에서 출력.

예시:

Time: 320, Child No.0 gets CPU time, Remaining:0, Run Queue: 1 2 3 4 6 7 9 IO Wait Queue: 0

Time: 321, Child No.1 gets CPU time, Remaining:11, Run Queue: 2 3 4 6 7 9 1 IO Wait Queue: 0

Time: 322, Child No.2 gets CPU time, Remaining:7, Run Queue: 3 4 6 7 9 1 2 IO Wait Queue:

Time: 323, Child No.3 gets CPU time, Remaining:0, Run Queue: 4 6 7 9 1 2 3 IO Wait Queue:

Time: 324, Child No.4 gets CPU time, Remaining:3, Run Queue: 6 7 9 1 2 3 4 IO Wait Queue:

Time: 325, Child No.6 gets CPU time, Remaining:4, Run Queue: 7 9 1 2 3 4 6 IO Wait Queue:

Time: 326, Child No.7 gets CPU time, Remaining:0, Run Queue: 9 1 2 3 4 6 IO Wait Queue: 7

Time: 327, Child No.9 gets CPU time, Remaining:6, Run Queue: 1 2 3 4 6 9 IO Wait Queue: 7

Time: 328, Child No.1 gets CPU time, Remaining:9, Run Queue: 2 3 4 6 9 1 IO Wait Queue:

Time: 329, Child No.2 gets CPU time, Remaining:5, Run Queue: 3 4 6 9 1 2 IO Wait Queue:

Time: 330, Child No.3 gets CPU time, Remaining:0, Run Queue: 4 6 9 1 2 IO Wait Queue:

1. **문제점, 막힌 부분 및 개선방안**
2. 문제점 및 막힌 부분: 처음에는 더 익숙한 Windows로 구현을 해보려고 했는데, Windows에는 메시지 큐 방식의 시스템 API를 제공하지 않았다. 그래서 비슷해 보이는 Mailslot 함수를 써보려고 했지만, Mailslot 함수는 단방향 통신만을 제공해서 부모와 자식 양쪽에 mailslot을 만들어 해보려 했지만 어려움이 있어서 ubuntu로 바꾸게 되었다. 바꾼 후 메시지 큐를 구현하려고 공부를 해보니, 1개의 메시지 큐로도 msg\_type을 다르게 하면 여러 프로세스들이 동시에 통신 가능하다고 해서 msgq를 1개만 생성하고 각 자식 프로세스들의 msg \_type을 2~11로 설정해 구현을 해보았다. 하지만 자식 프로세스가 msgrcv 함수를 통해 메시지를 읽어 들일 때, msgrcv의 네번 째 인자인 long msgtyp을 0이 아닌 다른 숫자를 넣으면 함수 실행이 제대로 되지 않는 현상이 발생했다. 어떤 부분이 잘못인지 찾지 못해서, 결국 각 자식 프로세스마다 다른 메시지 큐를 써서 메시지 큐들의 key\_id가 10개가 나오도록 바꿨다.
3. 개선방안: msgtyp을 통한 여러 프로세스들의 통신이 왜 안됐는지 파악을 해서 1개의 메시지 큐로도 통신이 되도록 해보아야겠다. 그리고 다른 방식의 스케줄링도 시도를 해보고 각 방법들의 장단점들은 뭐가 있는지 봐야겠다.

코드:

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <sys/stat.h>

#include <sys/time.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <malloc.h>

#include <signal.h>

#include <fcntl.h>

#define TIMEQUANTUM 5 //Real Timequantum: 1 sec

struct msgbuf

{

long msgtype; //parent 1, child 2~11

char mtext[256];

int data;

};

typedef struct node

{

int key;

pid\_t pid;

struct node \*prev;

struct node \*next;

}node;

typedef struct queue

{

node \*head;

node \*tail;

int count;

}queue;

typedef enum {true, false}bool;

int CPU\_burst\_time[10];

int IO\_burst\_time[10];

int ProcessNum;

int time\_count;

int child\_msg[10];

static int tick\_num = 0; //tick\_num 1 -> 0.5 sec

void message\_send(key\_t key\_id, struct msgbuf \*sndbuf);

int parent\_Receivemsg(int total\_child, struct msgbuf \*rcvbuffunc, key\_t key\_id);

bool isQueueEmpty(queue \*Queue);

//DoubleLinkedList Queue

node \*CreateNode(int key, pid\_t pid);

void InitQueue(queue \*Queue);

void Enqueue(queue \*Queue, node \*nod);

node \*Dequeue(queue \*Queue);

//buf for sprintf, write

char buf[256];

void timer\_handler(int signum)

{

tick\_num ++;

time\_count++;

}

int main()

{

//File open

int fd;

int written;

fd = open("/home/shin/Result.txt", O\_RDWR);

if(fd==-1)

{

printf("openError");

return 1;

}

// set msgq

key\_t key\_id[10];

struct msgbuf sndbuf, rcvbuf;

memset(sndbuf.mtext, 0x00, sizeof(sndbuf.mtext));

memset(rcvbuf.mtext, 0x00, sizeof(rcvbuf.mtext));

sndbuf.msgtype=2;

sndbuf.data =0;

rcvbuf.msgtype=2;

rcvbuf.data=0;

int k=0;

int i=0;

for(k=0;k<10;k++)

{

key\_id[k]= msgget((key\_t)(1234+k), IPC\_CREAT|0666);

if(key\_id[k] ==-1)

{

perror("msgget error");

exit(0);

}

}

//set timer

struct sigaction sa;

struct itimerval timer;

time\_count =0;

tick\_num =0;

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

sa.sa\_handler = &timer\_handler;

sigaction(SIGVTALRM, &sa, NULL);

timer.it\_value.tv\_sec =0;

timer.it\_value.tv\_usec = 100000\* TIMEQUANTUM;

timer.it\_interval.tv\_sec = 0;

timer.it\_interval.tv\_usec = 100000 \* TIMEQUANTUM;

setitimer(ITIMER\_VIRTUAL, &timer, NULL);

//child process fork

ProcessNum = 10;

int runningProcess = ProcessNum;

pid\_t pid[ProcessNum];

for(i=0;i<ProcessNum;i++)

{

CPU\_burst\_time[i] = rand()%20 + 60;

IO\_burst\_time[i] = rand()%10;

pid[i] = 50;

child\_msg[i]=0;

sprintf(buf, "Child No.%d, CPU Time:%d, IO Time:%d \n", i,CPU\_burst\_time[i],IO\_burst\_time[i]);

written = write(fd, buf, strlen(buf));

memset(buf, 0x00, 256);

}

for(i=0;i<ProcessNum;i++)

{

pid[i] = fork();

if(pid[i] == -1)

{

perror("fork error");

exit(1);

}

else if(pid[i] == 0)

{

//Child Process i

printf("Child Process No.%d PID: %d ", i, (int)getpid() );

printf("CPUTIME:%d \n", CPU\_burst\_time[i]);

struct msgbuf sndbufT, rcvbufT;

int Num = i;

memset(sndbufT.mtext, 0x00, sizeof(sndbuf.mtext));

sndbufT.data = Num;

sndbufT.msgtype = i+2;

rcvbufT.data =0;

rcvbufT.msgtype = i+2;

key\_t key\_idt;

key\_idt = msgget((1234+i), IPC\_CREAT|0666);

if(key\_idt<0)

{

perror("Child msgget error");

exit(0);

}

while(CPU\_burst\_time[Num]>0)

{

if(msgrcv( key\_idt, &rcvbufT, sizeof(struct msgbuf), 0, 0) == -1)

{

perror("msgrcv error");

exit(0);

}

CPU\_burst\_time[Num]-=2;

printf("\n Num: %d, left: %d \n", Num, CPU\_burst\_time[Num]);

fflush(stdout);

}

sndbufT.data = IO\_burst\_time[i];

message\_send(key\_id[i], &sndbufT);

printf("child IO");

fflush(stdout);

sleep(IO\_burst\_time[Num]+30);

printf("exit");

fflush(stdout);

exit(100+i);

}

}

//init readyqueue & waitqueue

queue readyQueue;

queue waitQueue;

InitQueue(&readyQueue);

InitQueue(&waitQueue);

for(i=0;i<ProcessNum;i++)

{

Enqueue(&readyQueue, CreateNode(CPU\_burst\_time[i], i));

}

//Schedule Start

node \*nod = NULL;

node \*nodT = NULL;

bool Q = true;

while(runningProcess>0)

{

if(time\_count == 2)

{

if((readyQueue.count)>0)

{

nod = Dequeue(&readyQueue);

sndbuf.data = (int)(nod->pid);

sndbuf.msgtype = (long)(sndbuf.data+2);

if(CPU\_burst\_time[sndbuf.data]>0)

{

//Send message to child & decrease CPU\_burst

message\_send(key\_id[sndbuf.data], &sndbuf);

CPU\_burst\_time[sndbuf.data]-=2;

printf("Parentcheck: %d, %d \n",sndbuf.data, CPU\_burst\_time[sndbuf.data]);

fflush(stdout);

time\_count =0;

Enqueue(&readyQueue, nod);

}

else

{

//If CPU\_burst is zero move to waitQueue and do I/O

CPU\_burst\_time[sndbuf.data]=0;

if(child\_msg[sndbuf.data]==0)

{

if(msgrcv(key\_id[sndbuf.data], &rcvbuf, sizeof(struct msgbuf), 0, 0)==-1)

{

perror("msgrcv error");

exit(0);

}

child\_msg[sndbuf.data]=1;

Enqueue(&waitQueue, nod);

printf("in IO");

fflush(stdout);

}

}

if(CPU\_burst\_time[sndbuf.data]<1) CPU\_burst\_time[sndbuf.data]=0;

sprintf(buf, "Time: %d, Child No.%d gets CPU time, Remaining:%d, Run Queue: ", tick\_num/2, sndbuf.data, CPU\_burst\_time[sndbuf.data]);

written = write(fd, buf, strlen(buf));

memset(buf, 0x00, 256);

for(i=0;i<readyQueue.count;i++)

{

nod = Dequeue(&readyQueue);

sprintf(buf, "%d ", nod->pid);

written = write(fd, buf, strlen(buf));

Enqueue(&readyQueue, nod);

}

}

else

{

sprintf(buf, "Time: %d, ",tick\_num/2);

written = write(fd, buf, strlen(buf));

memset(buf, 0x00, 256);

}

sprintf(buf, "IO Wait Queue: ");

written = write(fd, buf, strlen(buf));

memset(buf, 0x00, 256);

if(waitQueue.count>0)

{

//Schedule I/O queue

nodT=Dequeue(&waitQueue);

IO\_burst\_time[nodT->pid]-=2;

printf("\nIO: %d \n", IO\_burst\_time[nodT->pid]);

fflush(stdout);

if(IO\_burst\_time[nodT->pid]<1)

{

IO\_burst\_time[nodT->pid]=0;

child\_msg[nodT->pid]=2;

runningProcess--;

}

else Enqueue(&waitQueue, nodT);

for(i=0;i<waitQueue.count;i++)

{

nodT = Dequeue(&waitQueue);

sprintf(buf, "%d ", nodT->pid);

written = write(fd, buf, strlen(buf));

Enqueue(&waitQueue, nodT);

}

}

sprintf(buf, "\n");

written = write(fd, buf, strlen(buf));

memset(buf, 0x00, 256);

printf("\n runningPro: %d \n", runningProcess);

fflush(stdout);

time\_count =0;

}

}

printf("finish");

fflush(stdout);

close(fd);

return 0;

}

int parent\_Receivemsg(int total\_child, struct msgbuf \*rcvbuffunc, key\_t key\_id)

{

int num=1;

int check=10;

if(msgrcv(key\_id, &rcvbuffunc, sizeof(struct msgbuf), 0, 0)==-1)

{

perror("msgrcv error");

exit(0);

}

printf("num: %d", num);

fflush(stdout);

return num;

}

void message\_send(key\_t key\_id, struct msgbuf \*sndbuf)

{

if(msgsnd(key\_id, &sndbuf, sizeof(struct msgbuf), IPC\_NOWAIT) == -1)

{

perror("msgsnd error");

exit(0);

}

}

//Queue Functions

node \*CreateNode(int Key, pid\_t Pid)

{

node \*new = (node \*)malloc(sizeof(node));

new->key = Key;

new->pid = Pid;

new->prev = new->next = NULL;

return new;

}

void InitQueue(queue \*Queue)

{

Queue->head = CreateNode(0,0);

Queue->tail = CreateNode(0,0);

Queue->head->next = Queue->tail;

Queue->tail->prev = Queue->head;

Queue->count =0;

}

bool isQueueEmpty(queue \*Queue)

{

if(Queue->head->next = Queue->tail) return true;

else return false;

}

void Enqueue(queue \*Queue, node \*nod)

{

nod->prev = Queue->tail->prev;

nod->next = Queue->tail;

Queue->tail->prev->next = nod;

Queue->tail->prev = nod;

Queue->count++;

}

node \*Dequeue(queue \*Queue)

{

if(Queue->head->next != NULL)

{

node \*popnode = Queue->head->next;

popnode->prev->next = popnode->next;

popnode->next->prev = popnode->prev;

Queue->count--;

return popnode;

}

else return NULL;

}