**Постановка задачи**

Программа моделирует работу монитора обработки сообщений. Порожденные процессы, обладающие различными приоритетами и выполняющие некоторые циклы работ, посредством очереди сообщений передают родительскому процессу имена программ из предыдущих лабораторных работ, которые им должны быть запущены. Родительский процесс, обрабатывая сообщения в соответствии с их приоритетами, следит, чтобы одновременно было запущено не более трех программ.

**Описание метода решения задачи**

1. Родительский процесс создает очередь с ключом 10;
2. Если очередь с таким ключом существует, выдается сообщение об ошибке;
3. Родительский процесс создает набор семафоров с ключом 11;
4. Если набор с таким ключом существует, выдается сообщение об ошибке;
5. For i=0,5 do
6. Создаем дочерние процессы и помещаем имена файлов с определенным приоритетом в очередь;
7. Завершаем дочерние процессы;
8. Родительский процесс считывает из очереди имена файлов по приоритетам, запускает программы, отслеживая с помощью семафора, чтобы не было запущено больше 3-х.
9. Завершается работа после считывания всех файлов из очереди и завершения всех процессов.
10. Родительский процесс удаляет очередь и все данные в ней.
11. Родительский процесс удаляет набор семафоров.

**Описание программного средства**

Программа написана на языке C++. Для получения исполняемого файла исходный текст программы следует скомпилировать каким-либо компилятором языка C++. Исполняемый файл может работать под любой UNIX-совместимой ОС, поддерживающей соответствующий формат исполняемого файла.

Запуск программы осуществляется командой:

./lab7 ,

как видно, программа запускается без входных параметров. Если пользователь, вследствие своей глупости, либо в виду сложившихся обстоятельств запустит программу с какими-либо параметрами, она не отреагирует на данное недоразумение и продолжит работу вопреки всему. Сообщения о процессе работы программы выводятся на экран.

**Исходный текст**

Файл lab7.cpp

#include <sys/wait.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <sys/sem.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <iostream>

using namespace std;

sembuf sem\_buf; // for sem

/\* structure to work with the queue \*/

struct Msg

{

long mtype; // type of message (priority)

char mtext[10]; // text of message

};

/\*

\* Function of using resourse

\* @num - number of semaphores

\* @semid - id of semaphore

\*/

void p(int num,int semid)

{ sem\_buf.sem\_num=num;

sem\_buf.sem\_op=-1;

semop(semid,&sem\_buf,1);

}

/\*

\* Function of unusing resourse

\* @num - number of semaphores

\* @semid - id of semaphore

\*/

void v(int num,int semid)

{ sem\_buf.sem\_num=num;

sem\_buf.sem\_op=1;

semop(semid,&sem\_buf,1);

}

int main()

{

key\_t msgkey; // key of quene

key\_t semkey; // key of sem

int msgqid; // quene id

int semid; // sem id

Msg mm; // mm - message

int pid[6]; // id of processes

int \*stat\_loc, id; // for wait()

int n = 0;

char files[6][10] = {"lab\_1","lab\_2","lab\_3","lab\_4","lab\_5","lab\_6"};

msgkey = 10;

semkey = 11;

msgqid = msgget(msgkey,0666|IPC\_CREAT|IPC\_EXCL); // create new quene

semid = semget(semkey,1,0666|IPC\_CREAT|IPC\_EXCL); // create new sem

if (msgqid == -1) cout << "Error in creating new quene." << endl, exit(1);

if (semid == -1) cout << "Error in creating new semaphore." << endl, exit(1);

semctl(semid,0,SETVAL,3);

/\* first process \*/

switch(pid[0]=fork())

{

case -1:

cout << "Error in creating first process" << endl, exit(1);

case 0:

strncpy(mm.mtext,files[0],10); // set name of file

mm.mtype=5l; // set priority

if (msgsnd(msgqid,&mm,10,IPC\_NOWAIT)==-1) // send the queue name of the file

cout << "Error in sending 1 msg." <<endl, exit(1);

exit(0); break;

}

/\* second process \*/

switch(pid[1]=fork())

{

case -1:

cout << "Error in creating second process" << endl, exit(1);

case 0:

strncpy(mm.mtext,files[1],10); // set name of file

mm.mtype=4l; // set priority

if (msgsnd(msgqid,&mm,10,IPC\_NOWAIT)==-1) // send the queue name of the file

cout << "Error in sending 2 msg." <<endl, exit(1);

exit(0); break;

}

/\* third process \*/

switch(pid[2]=fork())

{

case -1:

cout << "Error in creating third process" << endl, exit(1);

case 0:

strncpy(mm.mtext,files[2],10); // set name of file

mm.mtype=6l; // set priority

if (msgsnd(msgqid,&mm,10,IPC\_NOWAIT)==-1) // send the queue name of the file

cout << "Error in sending 3 msg." <<endl, exit(1);

exit(0); break;

}

/\* fourth process \*/

switch(pid[3]=fork())

{

case -1:

cout << "Error in creating fourth process" << endl, exit(1);

case 0:

strncpy(mm.mtext,files[3],10); // set name of file

mm.mtype=1l; // set priority

if (msgsnd(msgqid,&mm,10,IPC\_NOWAIT)==-1) // send the queue name of the file

cout << "Error in sending 4 msg." <<endl, exit(1);

exit(0); break;

}

/\* fifth process \*/

switch(pid[4]=fork())

{

case -1:

cout << "Error in creating fifth process" << endl, exit(1);

case 0:

strncpy(mm.mtext,files[4],10); // set name of file

mm.mtype=2l; // set priority

if (msgsnd(msgqid,&mm,10,IPC\_NOWAIT)==-1) // send the queue name of the file

cout << "Error in sending 5 msg." <<endl, exit(1);

exit(0); break;

}

/\* sixth process \*/

switch(pid[5]=fork())

{

case -1:

cout << "Error in creating sixth process" << endl, exit(1);

case 0:

strncpy(mm.mtext,files[5],10); // set name of file

mm.mtype=3l; // set priority

if (msgsnd(msgqid,&mm,10,IPC\_NOWAIT)==-1) // send the queue name of the file

cout << "Error in sending 1 msg." <<endl, exit(1);

exit(0); break;

}

/\* waiting of the exiting of all processes \*/

for (int i = 0; i < 6; i++)

wait(stat\_loc);

cout << "Processes is completed." << endl;

/\* Reading messages on priority and starting labs\*/

for (long i = 6; i > 0; i--)

{

cout << semctl(semid,0,GETVAL,NULL) << endl;

msgrcv(msgqid,&mm,10,i,IPC\_NOWAIT|MSG\_NOERROR); // read from quene

cout << "Running '" << mm.mtext << "' with priority " << mm.mtype << ".\n";

switch(pid[i-1] = fork())

{

case -1:

cout << "Crash!" << endl, exit(1);

case 0:

p(0,semid); //sem\_val - 1

cout << "Working " << mm.mtext << "...\n" ;

sleep(1); // do lab

exit(0); break;

}

}

/\* waiting of the exiting of all processes \*/

for (int i = 0; i < 6; i++)

{

wait(stat\_loc);

cout << "Lab is completed." << endl;

v(0,semid); //sem\_val + 1

}

msgctl(msgqid,IPC\_RMID,NULL); // delete quene and all data in quene

semctl(semid,1,IPC\_RMID,NULL); // delete sem

}

**Пример выполнения**

Запуск: ./lab7

Вывод на экран:

Processes is completed.

Running 'lab\_3' with priority 6.

Running 'lab\_1' with priority 5.

Working lab\_3...

Running 'lab\_2' with priority 4.

Working lab\_1...

Lab is completed.

Running 'lab\_6' with priority 3.

Lab is completed.

Working lab\_2...

Running 'lab\_5' with priority 2.

Working lab\_6...

Lab is completed.

Running 'lab\_4' with priority 1.

Lab is completed.

Working lab\_5...

Working lab\_4...

Lab is completed.

Lab is completed.