**1.** **Цель работы**

Освоение средств IPC. Написание программ, использующих механизм семафоров, очередей сообщений, сегментов разделяемой памяти.

**2. Задача**

Родительский процесс помещает в сегмент разделяемой памяти имена программ из предыдущих лабораторных работ, которые могут быть запущены. Выполняя некоторые циклы работ, порожденные процессы случайным образом выбирают имена программ из таблицы сегмента разделяемой памяти, запускают эти программы, и продолжают свою работу. Посредством аппарата семафоров должно быть обеспечено, чтобы не были одновременно запущены две одинаковые программы. В процессе работы через очередь сообщений родительский процесс информируется, какие программы и от имени кого запущены.

**3. В программе были использованы системные вызовы:**

int **fork** () – порождение процесса-потомка

void **exit** (int status) – завершение функционирования процесса

int **wait** (int \*status) – ожидание завершения процесса-потомка

int **semget** (key\_t key, int count, int flag) – получение идентификатора набора семафоров

int **semctl** (int semid, int sem\_num, int command, union semun arg) – управление набором семафоров

int **msgget** (key\_t key, int flag) – получение идентификатора очереди сообщений

int **msgsnd** (int msgqid, void \*msg, size\_t size, int flag) – посылка сообщения через очередь

int **msgrcv** (int msgqid, void \*msg, size\_t size, long msg\_type, int flag) – получение сообщения из очередди

int **msgctl** (int msgqid, int command, struct msqid\_ds \*msg\_stat) – управление очередью сообщений

int **shmget** (key\_t key, size\_t size, int flag) – получение идентификатора сегмента разделяемой памяти

int **shmat** (int shmid, void \*daddr, int flags) – подключение сегмента к виртуальной памяти процесса

int **shmdt** (void \*daddr) – отключение сегмента от виртуальной памяти процесса

int **shmctl** (int shmid, int command, struct shmid\_ds \*shm\_stat) – управление сегментом разделяемой памяти

**4) Метод решения**

При выполнении программы основной процесс порождает два дочерних процесса. Основной процесс создаёт сегмент разделяемой памяти, набор из двух семафоров, очередь сообщений. В сегмент разделяемой памяти родительским процессом помещаются имена предыдущих лабораторных работ. Дочерние процессы случайным образом выбирают номер лабораторной работы, подлежащей исполнению, выполняет её и сообщает об этом родителю через очередь сообщений.

**5) Исходный код программы:**

*/\*\* @file upr7\_9.cpp*

*\* @brief Единственный файл приложения*

*\* @note Лабораторная работа N7 (вариант 9)*

*\* @author Чернов А. К. (shtyle@rambler.ru)*

*\* @date 2010*

*\* @par*

*\* Головной процесс несколько раз порождает потомков, каждый из которых*

*\* пытается запустить один из 2 исполлняемых файлов (./lab1, ./lab2).*

*\* Если программа уже запущениа, потомок дожидается своей очереди.*

*\* Имена программ передаются потомкам через область разделяемой памяти.*

*\* Информация о номере изменяемого семафора передается от потомка через*

*\* очередь сообщений.*

*\*/*

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/unistd.h>

#include <sys/wait.h>

#include <sys/ipc.h>

#include <sys/sem.h> *///< Семафоры*

#include <sys/msg.h> *///< Сообщения*

#include <sys/shm.h> *///< Разделяемая память*

*/\*\* Макрос - обработка некорректного завершения*

*\* @param x - возвращаемый код*

*\* @param y - возвращаемое сообщение*

*\*/*

#define MyError(int x,const char y[]) {\

perror(y);\

\_exit(x);\

}

*/\*\* Закрываем всё что открыли \*/*

#define CloseAll()\

{\

msgctl(msgfd,IPC\_RMID,0);\

semctl(semfd,IPC\_RMID,0);\

shmctl(shmfd,IPC\_RMID,0);\

}

*/\*\* Необходимые константы \*/*

enum {

MAGIC\_KEY = 0x011, *///< Для совместимости. Хоть какой-то ключ.*

MSG\_SIZE = 250, *///< Размер сообшения*

SHM\_SIZE = 1024, *///< Размер разделяемой памяти*

N\_SIZE = 10, *///< Длина имени вызываемых программы*

F\_NUM = 5, *///< Количество потомкков*

};

*/\*\* Определяем структуру семафора \*/*

union Semun {

int val;

struct semid\_ds \*buf;

unsigned short \*array;

};

*/\*\* Определяем структуру сообщения \*/*

struct Msgbuf {

long mtype;

char mtext[MSG\_SIZE];

};

int main(int argc, char \*argv[])

{

int semfd,msgfd,shmfd;

semfd = semget(MAGIC\_KEY,2, IPC\_CREAT);

if (semfd < 0) MyError(1,"semget error");

msgfd = msgget(MAGIC\_KEY,IPC\_CREAT);

if (msgfd < 0) MyError(2,"msgget error");

shmfd = shmget(MAGIC\_KEY,SHM\_SIZE,IPC\_CREAT | 0600);

if (shmfd < 0) {

msgctl(msgfd, IPC\_RMID,0);

MyError(3,"shmget error");}

char \*names;

if ((names = (char \*)shmat(shmfd,0,0)) == (char\*)-1) {

CloseAll();

MyError(3,"shmat error");}

Semun TmpOp;

TmpOp.val = 1;

semctl(semfd,0,SETVAL,TmpOp);

semctl(semfd,1,SETVAL,TmpOp);

sprintf((char\*)names,"lab1\0"); *///< Имена программ*

sprintf((char\*)(names+N\_SIZE),"cat lab7.cpp\0");

int p\_id;

srand((long)&TmpOp);

for (int i = 0; i < F\_NUM; i++)

{

switch(p\_id=fork())

{

case -1:

CloseAll();

MyError(4,"fork error");

break;

case 0:

struct sembuf tmp;

struct Msgbuf MyMsg;

int k = rand()&1;

tmp.sem\_num = k;

tmp.sem\_op = -1; *///< Пытаемся уменьшить семафор*

tmp.sem\_flg = 0;

semop(semfd,&tmp,sizeof(tmp));

sprintf(MyMsg.mtext,"%d\0",k); *///< Сообщаем номер программы*

msgsnd(msgfd,&MyMsg,sizeof(MyMsg),0);

char \*name = (char\*)shmat(shmfd,0,0)+k\*N\_SIZE;

execl(name,name,NULL);

exit(0);

}

}

int status;

struct sembuf tmp;

struct Msgbuf MyMsg;

int k;

tmp.sem\_op = 1;

tmp.sem\_flg = 0;

for (int i = 0; i < F\_NUM; i++)

{

wait(&status); *///< Ждем потомка*

msgrcv(msgfd,&MyMsg,sizeof(MyMsg),0,0); *///< Узнаем что он выполнял*

sscanf(MyMsg.mtext,"%d",&k);

tmp.sem\_num = k;

semop(semfd,&tmp,sizeof(tmp)); */// < Овобождаем семафор для следующих*

}

CloseAll();

return 0;

}