Министерство науки и образования РФ  
Новосибирский государственный технический университет  
Кафедра ТПИ

Лабораторная работа №7

по дисциплине «Управление ресурсами в вычислительных системах»

Факультет: ПМИ

Группа: ПМИ-31

Студенты: Суслов А.В.

Эльвейн Н.Л.  
 Олимпиади М.Б.

Преподаватели: Хайленко Е.А.  
 Филиппова Е.В.

Вариант: 1

1. **Цель работы**

Освоение средств IPC. Написание программ, использующих механизм

семафоров, очередей сообщений, сегментов разделяемой памяти.

1. **Задание**

Два дочерних процесса выполняют некоторые циклы работ, передавая после окончания очередного цикла через очередь сообщений родительскому процессу очередные четыре строки некоторого стихотворения, при этом первый процесс передает нечетные четырехстишья, второй - четные. Циклы работ процессов не сбалансированы по времени. Родительский процесс компонует из передаваемых фрагментов законченное стихотворение и выводит его по завершении работы обоих процессов. Решить задачу с использованием аппарата семафоров.

1. **Описание использованных системных вызовов**

**int semget(key\_t key, int nsems, int semflg) <sys/sem.h>**

Получает (или создаёт при выставленном флаге IPC\_CREAT) набор из nsems семафоров, привязанных к указанному системному ключу. Если в качестве ключа указано IPC\_PRIVATE, получить доступ к набору семафоров будет возможно только в рамках родственных процессов. Возвращает ID набора или -1 в случае неудачи.

**int semop(int semid, struct sembuf \*sops, size\_t nsops) <sys/sem.h>**

Совершает операцию sops.sem\_op над семафором sops.sem\_num из набора semid. Возможно обрабатывать массивы структур типа sembuf, в таком случае значение nsops будет равно количеству элементов (при обычном использовании nsops = 1).   
Возвращает -1 в случае неудачи, 0 в случае успеха.

**int msgget(key\_t key, int msgflg) <sys/msg.h>**

Получает (или создаёт при выставленном флаге IPC\_CREAT) очередь сообщений, привязанных к указанному системному ключу. Если в качестве ключа указано IPC\_PRIVATE, получить доступ к очереди будет возможно только в рамках родственных процессов.   
Возвращает ID очереди или -1 в случае неудачи.

**int msgsnd(int msqid, const void \*msgp, size\_t msgsz, int msgflg) <sys/msg.h>**

Отправляет сообщение msgp размером msgsz полезной части через очередь сообщений. Для корректной работы msgp должен быть указателем на структуру, в которой первое поле имеет тип long и означает тип передаваемого сообщения.   
Возвращает -1 в случае неудачи, 0 в случае успеха.

**ssize\_t msgrcv(int msqid, void \*msgp, size\_t msgsz, long msgtyp, int msgflg) <sys/msg.h>**

Выбирает следующее сообщение, подходящее по типу msgtyp, из очереди сообщений. Тип может принимать следующие значения: 0 - любой тип, >0 - первое вошедшее в очередь сообщение типа msgtyp, <0.   
Возвращает количество принятых байт или -1 в случае неудачи. Тип ssize\_t идентичен short.

1. **Текст программы**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h> // for errno

#include <string.h> // for strncpy()

#include <sys/wait.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/msg.h>

#pragma region Global vars

#define FORK\_ERROR 1 // Exit status for failed fork()

#define BUFSIZE 64 // I think that's enough for my input

#define MSGTYPE\_ANY 0

#define IPC\_NOFLAGS 0

#pragma endregion

typedef unsigned char byte; // this is missing in standard libs for whatever reason.

typedef struct { byte flags; byte code; } wait\_t; // union for wait result

typedef struct {

char \*command; // this is for compability with param\_start\_t

bool odd; // is process odd or not

int semid, mqid; // semaphore and message queue IDs

} param\_t; // Struct for child process execution data.

typedef struct {

long type; // this is for combatibility with POSIX syscalls

char text[BUFSIZE]; // useful data

} msg; // Struct for data to be sent via message queue.

const char \*filename = "poetry.txt"; // file with text

msg mbuf; // temporary store of message queue record

// Semaphore-specific actions for System V: A(S, N), D(S, N), Z(S)

#define sem\_a(sem\_id,sb,i,n) sb.sem\_num=(i); sb.sem\_op=(n); semop(sem\_id,&sb,1);

#define sem\_d(sem\_id,sb,i,n) sb.sem\_num=(i); sb.sem\_op=-(n); semop(sem\_id,&sb,1);

#define sem\_z(sem\_id,sb,i) sb.sem\_num=(i); sb.sem\_op=0; semop(sem\_id,&sb,1);

// Fill param\_t with child process execution data.

void create\_param(param\_t \*param, int sem\_id, int mq\_id, bool odd = true) {

param->command = "process\_poetry"; // this is safe to set -Wno-write-strings

param->semid = sem\_id;

param->mqid = mq\_id;

param->odd = odd;

}

// Create new thread in new process via fork() and run callback in it.

pid\_t fork\_thread(int(func(void \*param)), void \*param) {

int childpid;

switch (childpid = fork()) {

case -1:

fprintf(stderr, "[shell] Could not fork to subcommand '%s'\n", \*(char\*\*)param);

exit(FORK\_ERROR);

break;

case 0: exit(func(param)); /\* run callback and exit \*/ break;

default:

fprintf(stderr, "[shell] Process %d created for '%s'\n", childpid, \*(char\*\*)param);

return childpid;

break;

}

}

// Wait till ALL running subcommands end

int join\_threads() {

pid\_t wait\_pid;

wait\_t wait\_ret;

int exec\_result = 0;

while ((wait\_pid = wait(&wait\_ret)) > 0) {

printf("[shell] Process %d exited with code %d\n", wait\_pid, wait\_ret.code);

exec\_result |= wait\_ret.code; // OR all codes

}

return exec\_result;

}

// Callback function for thread to process input text file.

int process\_poetry(void \*\_param) {

param\_t \*param = (param\_t\*)\_param; // C style type conversion

FILE \*f = fopen(filename, "r"); // try to open poetry file

if (f == NULL)

return EXIT\_FAILURE; // immediately fail on non-existent file

char buf[BUFSIZE];

int num = 0; // current line number

struct sembuf sb; // semaphore control

sb.sem\_flg = IPC\_NOFLAGS; // flags are usually unset

mbuf.type = 1; // this MUST be positive, as said in man

char \*ptr = NULL;

do {

do {

num++; // increase current line number

ptr = fgets(buf, BUFSIZE - 1, f); // read line from file

if (ptr == NULL) break; // stop on eof

if ((num % 2) != param->odd) continue; // skip not owned lines

/\* используем два семафора. стартовое состояние: 0 1

первый процесс нечётный, второй чётный. 0 - доступно, 1 - заблокировано \*/

sem\_z(param->semid, sb, (short)param->odd); // wait till current is unblocked

// send message

strncpy(mbuf.text, buf, BUFSIZE - 1);

mbuf.text[strlen(buf) - 1] = '\0'; // remove line ending

if (msgsnd(param->mqid, &mbuf, BUFSIZE, IPC\_NOFLAGS) == -1) // try to send

return errno; // immediately fail on mq-errors

sem\_a(param->semid, sb, (short)param->odd, 1); // block itself

sem\_d(param->semid, sb, (short)!param->odd, 1); // unblock partner

} while ((num % 4) != 0); // break every 4 lines

fgets(buf, BUFSIZE - 1, f); // read separator line

} while (ptr != NULL); // stop on eof

return EXIT\_SUCCESS;

}

// Main entry point of application.

int main(int argc, char \*\*argv) {

key\_t mk = IPC\_PRIVATE; // use private key to simplify task

int mq = msgget(mk, IPC\_CREAT | 0600);

if (mq == -1) {

printf("Could not get message queue\n");

return EXIT\_FAILURE;

}

int sid = semget(mk, 2, IPC\_CREAT | 0600); // get two semaphores

if (sid == -1) {

printf("Could not get semaphore set\n");

return EXIT\_FAILURE;

}

struct sembuf sb;

sb.sem\_flg = IPC\_NOFLAGS; // don't use flags

sem\_a(sid, sb, 0, 1); // increase semaphore 0 value to 1 (init state: 0 1)

param\_t \*param = (param\_t\*)malloc(sizeof(param\_t)); // C style

create\_param(param, sid, mq, true); // init odd thread

fork\_thread(process\_poetry, (void\*)param); // run odd thread

create\_param(param, sid, mq, false); // init non-odd thread

fork\_thread(process\_poetry, (void\*)param); // run non-odd thread

join\_threads(); // wait for children to do all the hard work

ssize\_t cnt;

do {

cnt = msgrcv(mq, &mbuf, BUFSIZE, MSGTYPE\_ANY, IPC\_NOWAIT);

} while (cnt != -1 && printf("%s\n", mbuf.text));

return EXIT\_SUCCESS;

}

1. **Результат работы программы**

В файл **poetry.txt** был загружено стихотворение из 6 строф, по 4 строки в каждой. Каждый из дочерних процессов обрабатывал только "свои" (либо нечётные, либо нечётные строки), пропуская "чужие". На выходе имеем:

[shell] Process 1207 created for 'process\_poetry'

[shell] Process 1208 created for 'process\_poetry'

[shell] Process 1207 exited with code 0

[shell] Process 1208 exited with code 0

I liked it better when it didn't cost innocence

Lost myself in accomplishment

And all sight of all the things that I should see

And I say

Now the judge and the jury they sit and stare

On my own without you here

Just here me out

And take a look around

I'm letting go but all I see is evidence

Of what I know everything around just don't make sense

I'm letting go but all I feel is consequence

Of what I know everything around just don't make sense

I liked it better when I thought that I was free

One on one not two on me

This blind justice it seems to have a court

And I say

I feel it coming and I take it on myself

Before i succumb to someone else

Change the ending now

And let myself break free

I'm letting go but all I see is evidence

Of what I know everything around just don't make sense

I'm letting go but all feel is consequence

Of what I know everything around just don't make sense