|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**Лабораторная работа 2-я по Unix**

|  |  |
| --- | --- |
| **Тема: «Процессы. Системные вызовы fork() и exec()»**  **Студент** Гарасев Н.А.  **Группа** ИУ7-52Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Рязанова Н.Ю. |  |

Москва.

2020 г.

**Задание 1: написать программу, реализующую задачу «Производство-потребление» по алгоритму Э. Дейкстры с тремя семафорами: двумя считающими и одним бинарным.**

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <time.h>

#include <unistd.h>

#include <sys/wait.h>

#define COUNT\_PRODUCER 3

#define COUNCT\_CONSUMER 3

#define N 20

#define SEM\_BIN 0

#define SEM\_EMPTY 1

#define SEM\_FULL 2

#define P -1

#define V 1

struct sembuf producer\_start[2] = {

{SEM\_EMPTY, P, SEM\_UNDO},

{SEM\_BIN, P, SEM\_UNDO}

};

struct sembuf producer\_stop[2] = {

{SEM\_BIN, V, SEM\_UNDO},

{SEM\_FULL, V, SEM\_UNDO}

};

struct sembuf consumer\_start[2] = {

{SEM\_FULL, P, SEM\_UNDO},

{SEM\_BIN, P, SEM\_UNDO}

};

struct sembuf consumer\_stop[2] = {

{SEM\_BIN, V, SEM\_UNDO},

{SEM\_EMPTY, V, SEM\_UNDO}

};

#define PERMS S\_IRWXU | S\_IRWXG | S\_IRWXO

int sem\_id = -1;

int shm\_id = -1;

int \*shm = NULL;

int \*shm\_prod = NULL;

int \*shm\_cons = NULL;

void producer(const int id)

{

while(1)

{

sleep(1 + rand() % 5);

if (semop(sem\_id, producer\_start, 2) == -1)

{

perror("semop");

exit(1);

}

\*(shm + \*shm\_prod) = \*shm\_prod;

printf("Producer %d (pid %d) produces %d\n", id, getpid(), \*shm\_prod);

(\*shm\_prod)++;

if (semop(sem\_id, producer\_stop, 2) == -1) {

perror("semop");

exit(1);

}

}

}

void consumer(const int id)

{

while(1)

{

sleep(6 + rand() % 10);

if (semop(sem\_id, consumer\_start, 2) == -1) {

perror("semop");

exit(1);

}

printf("Consumer %d (pid %d) consumes %d\n", id, getpid(), \*(shm + \*shm\_cons));

(\*shm\_cons)++;

if (semop(sem\_id, consumer\_stop, 2) == -1) {

perror("semop");

exit(1);

}

}

}

/\* обработчик сигнала ctrl-c \*/

void catch\_sig(int sig\_numb)

{

signal(sig\_numb, catch\_sig);

shmctl(shm\_id, IPC\_RMID, NULL);

semctl(sem\_id, 0, IPC\_RMID, 0);

}

int main(void)

{

srand( time(0) );

sem\_id = semget(IPC\_PRIVATE, 3, IPC\_CREAT | PERMS);

if (sem\_id == -1) {

perror("semget");

exit(1);

}

if (semctl(sem\_id, SEM\_BIN, SETVAL, 1) == -1 ||

semctl(sem\_id, SEM\_EMPTY, SETVAL, N) == -1 ||

semctl(sem\_id, SEM\_FULL, SETVAL, 0) == -1) {

perror("semctl");

exit(1);

}

shm\_id = shmget(IPC\_PRIVATE, (N + 3) \* sizeof(int), IPC\_CREAT | PERMS);

if (shm\_id == -1) {

perror("shmget");

exit(1);

}

shm = shmat(shm\_id, 0, 0);

if (\*shm == -1) {

perror("shmat");

exit(1);

}

shm\_prod = shm;

shm\_cons = shm + 1;

\*shm\_prod = 0;

\*shm\_cons = 0;

shm = shm + 2;

for (int i = 0; i < COUNT\_PRODUCER; ++i)

{

const pid\_t pid = fork();

if (pid == -1)

{

perror("fork");

exit(1);

} else if (pid == 0)

{

producer(i);

exit(1);

}

}

for (int i = 0; i < COUNCT\_CONSUMER; ++i)

{

const pid\_t pid = fork();

if (pid == -1)

{

perror("fork");

exit(1);

} else if (pid == 0)

{

consumer(i);

exit(1);

}

}

signal(SIGINT, catch\_sig);

for (int i = 0; i < COUNT\_PRODUCER + COUNCT\_CONSUMER; ++i)

{

int status;

const pid\_t child\_pid = wait(&status);

if (child\_pid == -1) {

perror("wait");

exit(1);

}

if (WIFEXITED(status)) {

printf("Process %d returns %d\n", child\_pid, WEXITSTATUS(status));

} else if (WIFSIGNALED(status)) {

printf("Process %d terminated with signal %d\n", child\_pid, WTERMSIG(status));

} else if (WIFSTOPPED(status)) {

printf("Process %d stopped due signal %d\n", child\_pid, WSTOPSIG(status));

}

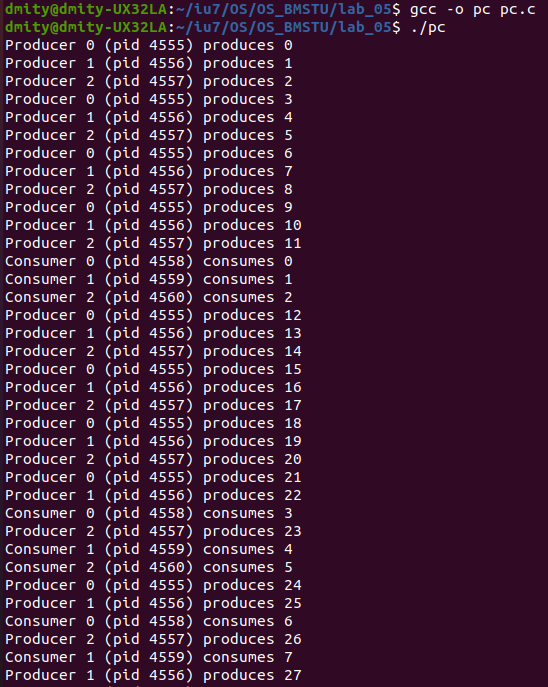
}

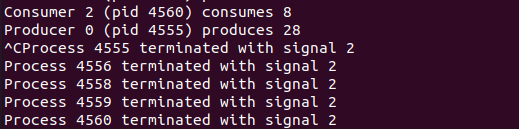
shmctl(shm\_id, IPC\_RMID, NULL);

semctl(sem\_id, SEM\_BIN, IPC\_RMID, 0);

return 0;

}

****

****

**Задание 2: написать программу, реализующую задачу «Читатели – писатели» по монитору Хоара с четырьмя функциями: Начать\_чтение, Закончить\_чтение, Начать\_запись, Закончить\_запись.**

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <time.h>

#include <unistd.h>

#include <sys/wait.h>

#define PERMS S\_IRWXU | S\_IRWXG | S\_IRWXO

#define COUNT 20

#define WRITERS 3

#define READERS 5

#define ACTIVE\_READERS 0 // активные читатели

#define WAIT\_WRITERS 2 // ждущие писатели

#define ACTIVE\_WRITERS 1 // активные писатели

#define WAIT\_READERS 3 // ждущие читатели

#define INC 1

#define DEC -1

struct sembuf start\_reading[] = {

{WAIT\_READERS, INC, SEM\_UNDO},

{WAIT\_WRITERS, 0, SEM\_UNDO},

{ACTIVE\_WRITERS, 0, SEM\_UNDO},

{WAIT\_READERS, DEC, SEM\_UNDO},

{ACTIVE\_READERS, INC, SEM\_UNDO}

};

struct sembuf stop\_read[] = {

{ACTIVE\_READERS, DEC, SEM\_UNDO}

};

struct sembuf start\_writing[] = {

{WAIT\_WRITERS, INC, SEM\_UNDO},

{ACTIVE\_WRITERS, 0, SEM\_UNDO},

{ACTIVE\_READERS, 0, SEM\_UNDO},

{WAIT\_WRITERS, DEC, SEM\_UNDO},

{ACTIVE\_WRITERS, INC, SEM\_UNDO}

};

struct sembuf stop\_write[] = {

{ACTIVE\_WRITERS, DEC, SEM\_UNDO}

};

int sem\_id = -1;

int shm\_id = -1;

int \*shm = NULL;

int \*shared\_value = NULL;

pid\_t\* child\_pids = NULL;

void writer(int number)

{

int start = semop(sem\_id, start\_writing, 5);

if (start == -1) {

perror("start writing");

exit(1);

}

(\*shared\_value)++;

printf("Writer #%d, pid=%d wrote value %d\n", number, getpid(), \*shared\_value);

int stop = semop(sem\_id, stop\_write, 1);

if (stop == -1) {

perror("stop write");

exit(1);

}

sleep(2);

}

void reader(int number)

{

int start = semop(sem\_id, start\_reading, 5);

if (start == -1) {

perror("start reading");

exit(1);

}

int val = \*shared\_value;

printf("Reader #%d, pid=%d read value: %d\n", number, getpid(), val);

int stop = semop(sem\_id, stop\_read, 1);

if (stop == -1) {

perror("stop read");

exit(1);

}

sleep(1);

}

/\* обработчик сигнала ctrl-c \*/

void catch\_sig(int sig\_numb)

{

signal(sig\_numb, catch\_sig);

shmctl(shm\_id, IPC\_RMID, NULL);

semctl(sem\_id, 0, IPC\_RMID, 0);

}

int main()

{

sem\_id = semget(IPC\_PRIVATE, 4, IPC\_CREAT | PERMS);

if (sem\_id == -1) {

perror("semget");

exit(1);

}

if (semctl(sem\_id, ACTIVE\_READERS, SETVAL, 0) == -1 ||

semctl(sem\_id, WAIT\_WRITERS, SETVAL, 0) == -1 ||

semctl(sem\_id, ACTIVE\_WRITERS, SETVAL, 0) == -1 ||

semctl(sem\_id, WAIT\_READERS, SETVAL, 0) == -1) {

perror("semctl");

exit(1);

}

shm\_id = shmget(IPC\_PRIVATE, sizeof(int), IPC\_CREAT | PERMS);

if (shm\_id == -1) {

perror("shmget");

exit(1);

}

shm = shmat(shm\_id, 0, 0);

if (\*shm == -1) {

perror("shmat");

exit(1);

}

shared\_value = shm;

child\_pids = shm + 1;

\*shared\_value = 0;

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

{

pid\_t pid;

if ((pid = fork()) == -1) {

printf("Can't fork");

exit(1);

}

if (pid == 0) {

printf("Writer #%d is running, pid: %d\n", i, getpid());

while (1) {

writer(i);

}

}

else {

child\_pids[i] = pid;

}

}

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

{

pid\_t pid;

if ((pid = fork()) == -1) {

printf("Can't fork");

exit(1);

}

if (pid == 0) {

printf("Reader #%d created, pid: %d\n", i, getpid());

while (1) {

reader(i);

}

}

else {

child\_pids[WRITERS + i] = pid;

}

}

signal(SIGINT, catch\_sig);

for (int i = 0; i < WRITERS + READERS; i++)

{

int status;

const pid\_t child\_pid = wait(&status);

if (child\_pid == -1) {

perror("wait");

exit(1);

}

if (WIFEXITED(status)) {

printf("Process %d returns %d\n", child\_pid, WEXITSTATUS(status));

} else if (WIFSIGNALED(status)) {

printf("Process %d terminated with signal %d\n", child\_pid, WTERMSIG(status));

} else if (WIFSTOPPED(status)) {

printf("Process %d stopped due signal %d\n", child\_pid, WSTOPSIG(status));

}

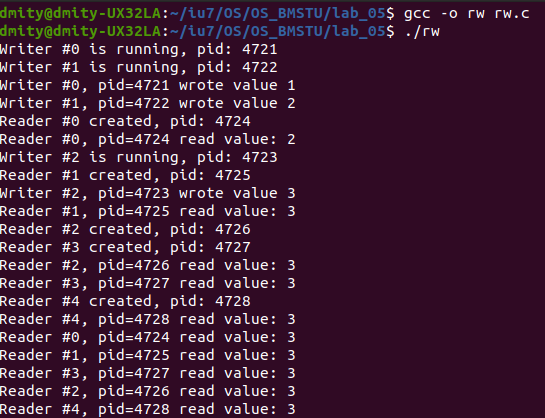
}

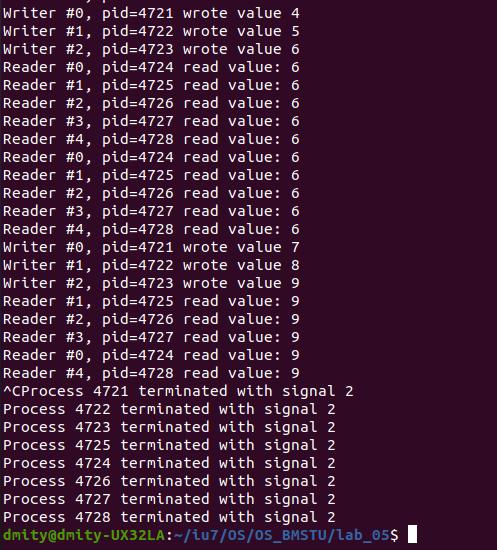
shmctl(shm\_id, IPC\_RMID, NULL);

semctl(sem\_id, 0, IPC\_RMID, 0);

return 0;

}

****

****