Постановка задачи:

Четыре дочерних процесса выполняют некоторые циклы работ, передавая после окончания очередного цикла через один и тот же сегмент разделяемой памяти родительскому процессу очередную строку некоторого стихотворения, при этом первый процесс передает 1-ю, 5-ю, 9-ю и т.д. строки, второй - 2-ю, 6-ю, 10-ю и т.д. строки, третий - 3-ю, 7-ю, 11-ю и т.д. строки, четвертый - 4-ю, 8-ю, 12-ю и т.д. строки. Циклы работ процессов не сбалансированы по времени. Родительский процесс компонует из передаваемых фрагментов законченное стихотворение и выводит его по завершении работы всех процессов. Решить задачу с использованием аппарата семафоров.

Описание метода решения задачи

Для организации работы четырех процессов в нужном режиме используется такое средство как набор семафоров. Он состоит из 6 семафоров. Первые четыре семафора соответствуют 4 процессам, пятый и шестой семафоры отвечают соответственно за запись и чтение из разделяемой памяти. Таким образом, процесс взаимодействия состоит в следующем. В начале работы родительский процесс устанавливает первый и пятый семафоры в 1 (разрешает операции), чтобы первый процесс мог начать запись, а все остальные сбрасывает. Когда первый процесс активизируется, то он сбрасывает соответствующий ему семафор и семафор записи в разделяемую память. Происходит запись строчки стихотворения в память. По окончании процесс устанавливает два семафора: для следующего процесса и для чтения из разделяемой памяти. Затем родительский процесс считывает данные из памяти и устанавливает семафор для записи. Каждый процесс сбрасывает свой семафор и устанавливает следующий. Четвертый процесс устанавливает первый семафор.

Описание программного средства

Для реализации поставленной задачи использованы следующие программные средства:

* int semget(key\_t key, int count, int flag) – для создания и получения доступа к набору семафоров;
* int semctl (int semid, int sem\_num, int command, union semun arg) – для удаления набора семафоров и других манипуляций;
* int semop (int semid, struct sembuf \*op\_array, count) - для изменения значений семафоров;
* int shmget (key\_t key, size\_t size, int flag) – для создания нового сегмента разделяемой памяти или нахождения уже существующего с тем же ключом;
* void \* shmat(int shmid, void \*daddr, int flags) - подключает сегмент с указанным описателем к виртуальной памяти обращающегося процесса;
* int shmdt(\*daddr) - отключает от виртуальной памяти ранее подключенный к ней сегмент с указанным виртуальным адресом начала; daddr - это виртуальный адрес начала сегмента в виртуальной памяти, ранее полученный от системного вызова shmat().

Исходный текст

/\*\*

\* Программа межпроцессного взаимодействия программ

\*

\*@author Buzmakova

\* @author Zubakova

\*@version 1.0

\*/

#include <iostream> /\*подключение стандартных библиотек для работы\*/

#include <sys/types.h>

#include <sys/sem.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <errno.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <wait.h>

using namespace std;

/\*\*

\* проверка корректности соответствующего системного вызова

\*

\*@ p результат системного вызова

\*@ key номер, соответствующий системному вызову, необходимый для

\*распознавания ошибки вызова

\*/

void proverka(int p, int key)

{

if(p < 0)

switch(key)

{

case 1:{ perror("oshibka SEMOP\n"); exit(1);}

case 2:{ perror("oshibka FORK\n"); exit(1);}

case 3:{ perror("oshibka SHMGET\n"); exit(1);}

case 4:{ perror("oshibka SHMDT\n"); exit(1);}

case 5:{ perror("oshibka SEMGET\n"); exit(1);}

case 6:{ perror("oshibka SEMGET\n"); exit(1);}

case 7:{ perror("oshibka SEMCTL\n"); exit(1);}

}

}

/\*\*

\* проверка корректности подключения сегмента к виртуальной памяти

\*

\*@ \*p результат системного вызова

\*/

void proverka\_1(void \* p)

{

if((char \*)p == (char \*)-1) { perror("oshibka SHMAT\n"); exit(1);}

}

union semun

{/\* тип, который используется для набора семафоров \*/

int val; /\*значение семафора\*/

struct semid\_ds \*mbuf; /\*информация о семафоре\*/

unsigned short \*array;

};

/\*\*

\* увеличивает значение семафора на 1

\*

\*@ semid идентификатор группы семафоров

\*@ sem номер семафора, к которому применяется функция

\*/

void v(int semid, int sem)

{

struct sembuf v\_buf; /\* содержит описание операций над семафорами \*/

/\* номер семафора, над которым будет произведена операция\*/

v\_buf.sem\_num = sem;

/\* число, которое будет прибавлено к значению семафора \*/

v\_buf.sem\_op = 1;

v\_buf.sem\_flg = 0;

/\* проверка корректности системного вызова и увеличение значения семафора\*/

proverka (semop (semid,&v\_buf,1), 1);

}

/\*\*

\* уменьшает значение семафора на 1

\*

\*@ semid идентификатор группы семафоров

\*@ sem номер семафора, к которому применяется функция

\*/

void p(int semid, int sem)

{

struct sembuf p\_buf; /\* содержит описание операций над семафорами \*/

/\* номер семафора, над которым будет произведена операция\*/

p\_buf.sem\_num = sem;

/\* число, которое будет вычтено из значения семафора \*/

p\_buf.sem\_op = -1;

p\_buf.sem\_flg = 0;

/\* проверка корректности системного вызова и уменьшение значения семафора\*/

proverka (semop (semid,&p\_buf,1), 1);

}

/\*\*

\* создание и инициализация набора семафоров

\*

\*@return semid идентификатор набора семафоров

\*/

int initsem()

{

int status = 0, semid; /\* идентификатор группы семафоров\*/

/\* струтура для хранения необходимой для управления нобором семафоров информации \*/

union semun arg;

arg.val = 0; /\* начальное значение набора семафоров \*/

/\* Создаём набор из 6 семафоров и сообщаем об ошибке в случае неудачи (и завершаем процесс)\*/

proverka((semid = semget(IPC\_PRIVATE,6,0600 | IPC\_CREAT | IPC\_EXCL)), 5);

/\* инициализация набора семафоров нулевыми значениями и проверка корректности вызова semctl \*/

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

proverka(semctl(semid,i,SETVAL,arg), 7);

/\*устанавливаем 1 и 5семафоры, чтобы первый процесс мог начать запись\*/

v(semid,0,0);

v(semid,4,0);

return semid;

}

/\*\*

\* запись строчек стихотворения дочерними процессами

\*

\*@ semid идентификатор группы семафоров

\*@ str набор строк стихотворения для этого процесса

\*@ shmid идентификатор разделяемой области памяти

\*@ number номер процесса

\*/

void write\_(int semid, char \*\*str, int shmid, int number)

{

char \*adrr; /\*указатель на разделяемую область памяти\*/

int j = number + 1; /\* номер следующего процесса\*/

if(number == 3) j = 0; /\*4 процесс устанавливает первый семафор\*/

/\*Подключение разделяемой области памяти и проверка корректности вызова shmat\*/

proverka\_1 ((adrr = (char \*)shmat(shmid,0,0)));

/\* Каждый процесс выводит в разделяемую область памяти по три строчки\*/

for (int i = 0; i < 3; i++)

{

p(semid, number,0); /\*сбрасываем семафор текущего процесса\*/

p(semid, 4,0); /\* сбрасываем семафор записи \*/

strcpy (adrr, str[i]); /\* запись строчки в разделяемую память \*/

v(semid, j,0); \*установка семафора для следующего процесса\*/

v(semid, 5,0); /\*установка семафора для чтения \*/

}

proverka(shmdt(adrr), 4); /\*отключение разделяемой памяти и проверка корректности вызова shmdt \*/

exit(0);

}

/\*\*

\* запуск процессов-потомков

\*

\*@ shmid идентификатор разделяемой области памяти

\*@ str набор строк стихотворения для этого процесса

\*@ num номер процесса

\*@ semid идентификатор группы семафоров

\*@return f идентификатор процесса-потомка

\*/

int fork\_(int shmid, char \*\*str, int num, int semid)

{

int f;

proverka((f = fork()), 2); /\* проверка коррексности вызова fork() \*/

if (f == 0) /\* если нет ошибки \*/

{

write\_(semid, str, shmid, num); /\* запись строчки в разделяемую память \*/

}

return f;

}

/\*\*

\* чтение строчек стихотворения родительским процессом

\*

\*@ semid идентификатор группы семафоров

\*@ shmid идентификатор разделяемой области памяти

\*@ \*addr указатель на разделяемую область памяти

\*/

void read\_(int semid, int shmid, char \*addr)

{

char str1[90];

p(semid, 5,shmid); /\* сбрасываем семафор, разрешающий чтение \*/

strcpy (str1,addr); /\* считываем данные из разделяемого сегмента \*/

cout << str1; /\* выводим строчку \*/

memset(addr,0,sizeof(addr)); /\* очищаем разделяемую область памяти\*/

v(semid, 4,shmid); /\* увеличиваем 5 семафор(разрешение записи) \*/

}

int main ()

{

/\*semid - идентификатор группы семафоров,shmid - идентификатор разделяемой области памяти,pr1,pr2,pr3,pr4 - идентификаторы порождённых процессов\*/

int shmid, semid,pr1,pr2,pr3,pr4;

char \*addr, str1[90];

char \*str[4][3]; /\*str\_proc - строки, которые передаются в порождённые процессы\*/

str[0][0] = "Мы все учились понемногу\n";

str[1][0] = "Чему-нибудь и как-нибудь,\n";

str[2][0] = "Так воспитаньем, слава богу,\n";

str[3][0] = "У нас немудрено блеснуть.\n";

str[0][1] = "Онегин был по мненью многих\n";

str[1][1] = "(Судей решительных и строгих)\n";

str[2][1] = "Ученый малый, но педант:\n";

str[3][1] = "Имел он счастливый талант\n";

str[0][2] = "Без принужденья в разговоре\n";

str[1][2] = "Коснуться до всего слегка,\n";

str[2][2] = "С ученым видом знатока\n";

str[3][2] = "Хранить молчанье в важном споре.\n";

semid=initsem(); /\*инициализация семафоров \*/

proverka((shmid=shmget(IPC\_PRIVATE,1024,IPC\_CREAT|S\_IRWXU|S\_IRWXG

|S\_IRWXO)), 3); /\*инициализируем разделяемую область памяти\*/

proverka\_1((addr=(char \*) shmat(shmid,0,0))); /\*подключение разделяемой памяти\*/

/\*порождение 4 процессов-потомков \*/

pr1 = fork\_(shmid,str[0],0,semid);

pr2 = fork\_(shmid,str[1],1,semid);

pr3 = fork\_(shmid,str[2],2,semid);

pr4 = fork\_(shmid,str[3],3,semid);

for (int i = 0; i < 12; i++)

{ /\* чтение 12 строчек \*/

read\_(semid, shmid, addr);

}

proverka(shmdt(addr), 4); /\*отключение разделяемой памяти и проверка корректности вызова shmdt \*/

waitpid(pr1, NULL, 0); /\* ожидание завершения 4 потомков\*/

waitpid(pr1, NULL, 0);

waitpid(pr1, NULL, 0);

waitpid(pr1, NULL, 0);

semctl(semid,0,IPC\_RMID,NULL); /\* удаление набора семафоров \*/

return (0); /\* успешное завершение \*/

}

**Примеры выполнения**

Во всех тестах входные данные отсутствуют

**Тест 1:**

**Запуск программы:**

./labka7

**Результат выполнения:**

Мы все учились понемногу

Чему-нибудь и как-нибудь,

Так воспитаньем, слава богу,

У нас немудрено блеснуть.

Онегин был по мненью многих

(Судей решительных и строгих)

Ученый малый, но педант:

Имел он счастливый талант

Без принужденья в разговоре

Коснуться до всего слегка,

С ученым видом знатока

Хранить молчанье в важном споре.

**Тест 2:**

**Запуск программы:**

./labka7

**Результат выполнения:**

oshibka SEMOP

**Тест 3:**

**Запуск программы:**

./labka7

**Результат выполнения:**

oshibka FORK

**Тест 4:**

**Запуск программы:**

./labka7

**Результат выполнения:**

oshibka SHMGET

**Тест 5:**

**Запуск программы:**

./labka7

**Результат выполнения:**

oshibka SHMDT

**Тест 6:**

**Запуск программы:**

./labka7

**Результат выполнения:**

oshibka SEMGET

**Тест 7:**

**Запуск программы:**

./labka7

**Результат выполнения:**

oshibka SEMGET

**Тест 8:**

**Запуск программы:**

./labka7

**Результат выполнения:**

oshibka SEMCTL

**Тест 9:**

**Запуск программы:**

./labka7

**Результат выполнения:**

oshibka SHMAT