rМосковский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовой проект
«Операционные системы»

Студент: Болдырев Антон Константинович
Группа: М8О–206Б–18
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2020.

Содержание

Общие сведения о программе

Необходимо написать 3-и программы. Далее будем обозначать их программы A, B, C соответственно. A принимает из стандартного потока ввода строки, а далее их отправляет программе C. Отправка строк должна производиться построчно. Программа C печатает в стандартный вывод полученную строку от программы A. После получения программа C отправляет программе A сообщение о том, что строка получена. До тех пор, пока строка A не примет «сообщение о получении строки» от программы C, она не может отправлять следующую строку программе C.
Программа B пишет в стандартный поток вывода количество отправленных символов программой A и количество принятых символов программой C. Данную информацию программа B получает от программ A и C соответственно.

Метод решения и алгоритм

Метод решения довольно прост: использовать разделяемую память, отображение файла в память и семафоры.

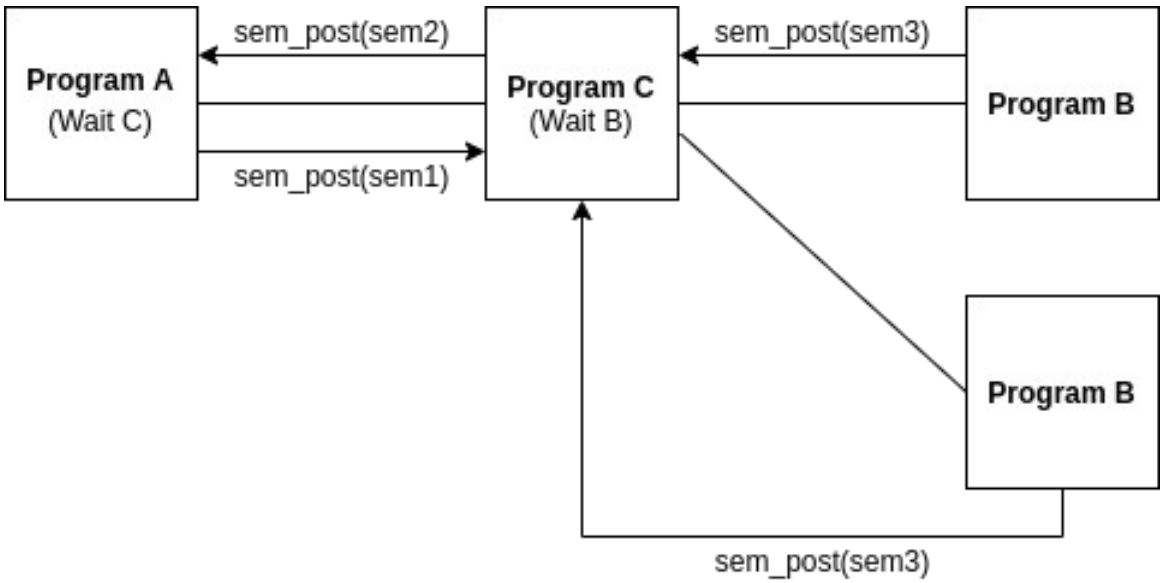
Алгоритм заключается в следующем:
1. Программа A создает необходимые семафоры, объект разделяемой памяти, отображает 100 байт в память. Считывает строку со стандартного потока, затем выполняет системный вызов создает дочерний процесс.

2. Дочерний процесс в свою очередь создает еще один процесс, который запускает программу B, для подсчета длины строки, переданной программой A.

3. После выполнения процесса B, выполняется процесс C, в котором тоже присутствует системный вызов для создания дочернего процесса B.

4. Процесс B считает длину строки, полученной программой C.

5. Процесс A ждет завершения дочерних процессов, после чего считывает новую строку со стандартного потока.



Основные файлы

a.c:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <semaphore.h>
#include <sys/mman.h>
#define BUF_SIZE 100
#define SHARED_MEMORY_NAME "/shm_file"
#define FIRST_SEM "/sem1"
#define SECOND_SEM "/sem2"
#define THIRD_SEM "/sem3"
int main()
{
   int fd_shm;
   char* shmem;
   char* tmp = (char*)malloc(sizeof(char) * BUF_SIZE);
   char* buf_size = (char*)malloc(sizeof(char) * 10);
   sem_t* sem1 = sem_open(FIRST_SEM, O_CREAT, 0660, 0);
   sem_t* sem2 = sem_open(SECOND_SEM, O_CREAT, 0660, 0);
   sem_t* sem3 = sem_open(THIRD_SEM, O_CREAT, 0660, 0);
   if (sem1 == SEM_FAILED || sem2 == SEM_FAILED || sem3 == SEM_FAILED) {
      perror("Semaphore opening error, program 'a'\n");
      exit(1);
   }
   if (shm_unlink(SHARED_MEMORY_NAME) == -1) {
      perror("shm_unlink error\n");
      exit(1);
   }
   // Get shared memory obj
   if ((fd_shm = shm_open(SHARED_MEMORY_NAME, O_RDWR | O_CREAT | O_EXCL, 0660))
== -1) {
      perror("shm_open error, program 'a'\n");
      exit(1);
   }
   // Allocate memory for shm obj
   if (ftruncate(fd_shm, BUF_SIZE) == -1) {
      perror("ftruncate error, program 'a'\n");
      exit(-1);
   }
   // Create a new mapping
   shmem = (char*)mmap(NULL, BUF_SIZE, PROT_WRITE | PROT_READ, MAP_SHARED,
fd_shm, 0);
   //convert file descriptor to string
   sprintf(buf_size, "%d", BUF_SIZE);
   char* argv[] = { buf_size, SHARED_MEMORY_NAME, SECOND_SEM, THIRD_SEM, NULL };
   while (scanf ("%s", tmp)) {
      pid_t p = fork();
      if (p == 0) {
            pid_t p_1 = fork();
            if (p_1 == 0) {
               sem_wait(sem1);
               printf("program a sent:\n");
```

```
            if (execve("./b.out", argv, NULL) == -1) {
               perror("Could not execve, program 'a'\n");
            }
         } else if (p_1 > 0) {
            sem_wait(sem3);
            if (execve("./c.out", argv, NULL) == -1) {
               perror("Could not execve, program 'a'\n");
            }
         }
      } else if (p > 0) {
         sprintf(shmem, "%s", tmp);
         sem_post(sem1);
         sem_wait(sem2);
         printf("###############\n\n");
      }
   }
   shm_unlink(SHARED_MEMORY_NAME);
   sem_unlink(FIRST_SEM);
   sem_unlink(SECOND_SEM);
   sem_unlink(THIRD_SEM);
   sem_close(sem1);
   sem_close(sem2);
   sem_close(sem3);
   close(fd_shm);
}
```

b.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <semaphore.h>
#include <sys/mman.h>
int main(int argc,  char const * argv[]) {
   if (argc < 2) {
      perror("not too much arg, program 'b'\n");
      exit(1);
   }
   int buf_size = atoi(argv[0]);
   char const* shared_memory_name = argv[1];
   char const* sem3_name = argv[3];
   int fd_shm;
   if ((fd_shm = shm_open(shared_memory_name, O_RDWR , 0660)) == -1) {
      perror("shm_open error, program 'b'\n");
      exit(1);
   }
   sem_t* sem3 = sem_open(sem3_name, 0,0,0);
   if (sem3 == SEM_FAILED) {
      perror("sem3 error, program 'b'\n");
      exit(1);
   }
```

```
   char* shmem = (char*)mmap(NULL, buf_size, PROT_WRITE | PROT_READ, MAP_SHARED,
fd_shm, 0);
   int size = strlen(shmem);
   printf("%d symbols\n", size);
   sem_post(sem3);
}
```

c.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <semaphore.h>
#include <sys/mman.h>
int  main(int argc, char* const argv[])
{
   if (argc < 2) {
      printf("not to much arg, program 'c'\n");
      return 0;
   }
   int buf_size = atoi(argv[0]);
   char const* shared_memory_name = argv[1];
   char const* sem2_name = argv[2];
   char const* sem3_name = argv[3];
   int fd_shm;
   if ((fd_shm = shm_open(shared_memory_name, O_RDWR , 0660)) == -1) {
      perror("shm_open error, program 'c'\n");
      exit(1);
   }
   sem_t* sem2 = sem_open(sem2_name, 0,0,0);
   sem_t* sem3 = sem_open(sem3_name, 0,0,0);
   if (sem2 == SEM_FAILED || sem3 == SEM_FAILED) {
      perror("sem2 error, program 'c'\n");
      exit(1);
   }
   char* shmem = (char*)mmap(NULL, buf_size, PROT_WRITE | PROT_READ, MAP_SHARED,
fd_shm, 0);
   pid_t p = fork();
   if (p == 0) {
      printf("program c take:\n");
      if (execve("b.out", argv, NULL) == -1) {
         perror("execve error, program 'c'\n");
         exit(1);
      }
   } else if (p > 0) {
      sem_wait(sem3);
      printf("%s\n", shmem);
   }
   sem_post(sem2);
}
```

makefile:

```
KEYS=-lrt -lpthread
all: a.c c.c
   gcc a.c -o a.out $(KEYS)
   gcc c.c -o c.out $(KEYS)
   gcc b.c -o b.out $(KEYS)
a: a.c
   gcc a.c -o a.out $(KEYS)
b: b.c
   gcc b.c -o b.out $(KEYS)
c: c.c
   gcc c.c -o c.out $(KEYS)
```

Пример работы

anton@anton-Lenovo-ideapad-320-15IKB:~/Education/OS/KP$ ./a.out
anton
program a sent:
5 symbols
program c take:
5 symbols
anton
################

boldyrev
program a sent:
8 symbols
program c take:
8 symbols
boldyrev

Вывод

Выполнив курсовой проект я написал многопроцессорную программу, используя средства, изученные в курсе «операционные системы». Сложность вызвало именно идея, представление, то как реализовать задание, но когда понимание пришло задача показалась простой. Отображение файлов в память и разделяемая память, по моему мнению могут быть применимы в большом ряде задач для многопоточных программ.