Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовой проект по курсу
«Операционные системы»

Студент: Ивченко Анна Владимировна
Группа: М8О-208Б-20
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Содержание

Репозиторий

https://github.com/Anetta123/OS/tree/main/kp

Постановка задачи

Необходимо написать 3 программы. Далее будем обозначать эти программы A, B, C. Программа A принимает из стандартного потока ввода строки, а далее их отправляет программе C. Отправка строк должна производится 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 программа A не примет «сообщение о получение строки» от программы C, она не может отправлять следующую строку программе C. Программа B пишет в стандартный вывод количество отправленных символов программой A и количество принятых символов программой C. Данную информацию программа B получает от программ A и C соответственно. Способ организация межпроцессорного взаимодействия выбирает студент.

Общие сведения о программе: программа состоит из трех файлов: A.cpp, B.cpp, C.cpp.

Общий метод и алгоритм решения: В начале работы создаются два дочерних процесса для B и C, а родительский процесс замещается программой A с помощью execl, сначала A с помощью getline считывает строку, передаёт в B количество считанных символов, а в C — количество считанных символов и саму строку посимвольно, затем B выводит количество введённых символов, C выводит строку и передаёт B количество выведенных символов, после чего B выводит количество выведенных символов и цикл начинается заново. Межпроцессорное взаимодействие основано на семафорах и pipe.

Исходный код:

A.cpp

```
#include <iostream>
    #include <unistd.h>
    #include <fcntl.h>
    #include <semaphore.h>
    using namespace std;
    int quick_get(sem_t *semaphore){
      int s;
      sem_getvalue(semaphore, &s);
      return s;
    }
    void quick_set(sem_t *semaphore, int n){
      while (quick_get(semaphore) < n){
        sem_post(semaphore);
      }
      while (quick_get(semaphore) > n){
        sem_wait(semaphore);
      }
```

```
}
int main(){
    int fdAC[2];
    int fdAB[2];
    int fdBC[2];
    pipe(fdAC);
    pipe(fdAB);
    pipe(fdBC);
    sem_unlink("_semA");
    sem_unlink("_semB");
    sem_unlink("_semC");
    sem_t* semA = sem_open("_semA", O_CREAT, 0751, 1);
    sem_t* semB = sem_open("_semB", O_CREAT, 0751, 0);
    sem_t* semC = sem_open("_semC", O_CREAT, 0751, 0);
    if ((semA == SEM_FAILED)||(semB == SEM_FAILED)||(semC == SEM_FAILED)){
        perror("sem_open");
        return -1;
    }
    cout << "Enter strings:\n";
    pid_t C = fork();
    if (C == -1){
        perror("fork");
        return -1;
    }
    if (C == 0){
        pid_t B = fork();
        if (B == -1){
            perror("fork");
            return -1;
        }
        if (B == 0){
            execl("B", to_string(fdAB[0]).c_str(), to_string(fdBC[0]).c_str(), NULL);
        }
        else{
            execl("C", to_string(fdAC[0]).c_str(), to_string(fdBC[1]).c_str(), NULL);
        }
    }
    else{
       while(1){
        string str;
        getline(cin, str);
        if (!cin.good()){
            quick_set(semA, 2);
            quick_set(semB, 2);
            quick_set(semC, 2);
            break;
        }
        int size = str.length();
```

```
            write(fdAB[1], &size, sizeof(int));
            write(fdAC[1], &size, sizeof(int)); //fd[0]-read, fd[1]-write
            for (int i = 0; i < size; ++i){
                write(fdAC[1], &str[i], sizeof(char));
            }
            quick_set(semB, 1);
            quick_set(semA, 0);
            while (quick_get(semA) == 0){
                continue;
            }
        } }
        sem_close(semA);
        sem_destroy(semA);
        sem_close(semB);
        sem_destroy(semB);
        sem_close(semC);
        sem_destroy(semC);
        close(fdAC[0]);
        close(fdAC[1]);
        close(fdAB[0]);
        close(fdAB[1]);
        return 0;
    }
```

B.cpp

```
#include <iostream>
#include <unistd.h>
#include <fcntl.h>
#include <semaphore.h>
using namespace std;


int quick_get(sem_t *semaphore){
    int s;
    sem_getvalue(semaphore, &s);
    return s;
}
void quick_set(sem_t *semaphore, int n){
    while (quick_get(semaphore) < n){
        sem_post(semaphore);
    }
    while (quick_get(semaphore) > n){
        sem_wait(semaphore);
    }
}

int main(){
    int fdAC[2];
    int fdAB[2];
    int fdBC[2];
    pipe(fdAC);
```

```
pipe(fdAB);
pipe(fdBC);
sem_unlink("_semA");
sem_unlink("_semB");
sem_unlink("_semC");
sem_t* semA = sem_open("_semA", O_CREAT, 0751, 1);
sem_t* semB = sem_open("_semB", O_CREAT, 0751, 0);
sem_t* semC = sem_open("_semC", O_CREAT, 0751, 0);
if ((semA == SEM_FAILED)||(semB == SEM_FAILED)||(semC == SEM_FAILED)){
    perror("sem_open");
    return -1;
}
cout << "Enter strings:\n";
pid_t C = fork();
if (C == -1){
    perror("fork");
    return -1;
}
if (C == 0){
    pid_t B = fork();
    if (B == -1){
        perror("fork");
        return -1;
    }
    if (B == 0){
        execl("B", to_string(fdAB[0]).c_str(), to_string(fdBC[0]).c_str(), NULL);
    }
    else{
        execl("C", to_string(fdAC[0]).c_str(), to_string(fdBC[1]).c_str(), NULL);
    }
}
else{
   while(1){
    string str;
    getline(cin, str);
    if (!cin.good()){
        quick_set(semA, 2);
        quick_set(semB, 2);
        quick_set(semC, 2);
        break;
    }
    int size = str.length();
    write(fdAB[1], &size, sizeof(int));
    write(fdAC[1], &size, sizeof(int)); //fd[0]-read, fd[1]-write
    for (int i = 0; i < size; ++i){
        write(fdAC[1], &str[i], sizeof(char));
    }
    quick_set(semB, 1);
    quick_set(semA, 0);
    while (quick_get(semA) == 0){
        continue;
    }
} }
sem_close(semA);
sem_destroy(semA);
sem_close(semB);
```

```
    sem_destroy(semB);
    sem_close(semC);
    sem_destroy(semC);
    close(fdAC[0]);
    close(fdAC[1]);
    close(fdAB[0]);
    close(fdAB[1]);
    return 0;
}#include <iostream>
#include <unistd.h>
#include <fcntl.h>
#include <semaphore.h>
using namespace std;

int quick_get(sem_t *semaphore){
    int s;
    sem_getvalue(semaphore, &s);
    return s;
}

void quick_set(sem_t *semaphore, int n){
    while (quick_get(semaphore) < n)
    {
        sem_post(semaphore);
    }
    while (quick_get(semaphore) > n)
    {
        sem_wait(semaphore);
    }
}

int main(int args, char* argv[]){
    int fdAB[1];
    fdAB[0] = atoi(argv[0]);
    int fdBC[1];
    fdBC[0] = atoi(argv[1]);
    sem_t* semA = sem_open("_semA", O_CREAT, 0751, 1);
    sem_t* semB = sem_open("_semB", O_CREAT, 0751, 0);
    sem_t* semC = sem_open("_semC", O_CREAT, 0751, 0);
    while (1){
        while(quick_get(semB) == 0){
            continue;
        }
        if (quick_get(semB) == 2){
            break;
        }
        int size;
        read(fdAB[0], &size, sizeof(int));
        cout << "Number of input symbols is " << size << endl;
        quick_set(semC, 1);
        quick_set(semB, 0);
        while (quick_get(semB) == 0){
            continue;
        }
        if (quick_get(semB) == 2){
            break;
```

```
        }
        read(fdBC[0], &size, sizeof(int));
        cout << "Number of output symbols is " << size << endl;
        quick_set(semA, 1);
        quick_set(semB, 0);
        while(quick_get(semB) == 0){
            continue;
        }
        if (quick_get(semB) == 2){
            break;
        }
    }
    sem_close(semA);
    sem_close(semB);
    sem_close(semC);
    close(fdAB[0]);
    close(fdBC[0]);
    return 0;
}
```

C.cpp

```
#include <iostream>
#include <unistd.h>
#include <fcntl.h>
#include <semaphore.h>
using namespace std;

int quick_get(sem_t *semaphore){
    int s;
    sem_getvalue(semaphore, &s);
    return s;
}

void quick_set(sem_t *semaphore, int n){
    while (quick_get(semaphore) < n){
        sem_post(semaphore);
    }
    while (quick_get(semaphore) > n){
        sem_wait(semaphore);
    }
}

int main(int args, char* argv[]){
    int fdAC[1];
    fdAC[0] = atoi(argv[0]);
    int fdBC[2];
```

```
    fdBC[1] = atoi(argv[1]);
    sem_t* semA = sem_open("_semA", O_CREAT, 0751, 1);
    sem_t* semB = sem_open("_semB", O_CREAT, 0751, 0);
    sem_t* semC = sem_open("_semC", O_CREAT, 0751, 0);
    while(1){
        while(quick_get(semC) == 0){
            continue;
        }
        if (quick_get(semC) == 2){
            break;
        }
        int size;
        string str;
        read(fdAC[0], &size, sizeof(int));
        int t = 0;
        for (int i = 0; i < size; ++i){
            char c;
            read(fdAC[0], &c, sizeof(char));
            str.push_back(c);
            t = i;
        }
        ++t;
        cout << str << endl;
        write(fdBC[1], &t, sizeof(int));
        quick_set(semB, 1);
        quick_set(semC, 0);
    }
    sem_close(semA);
    sem_close(semB);
    sem_close(semC);
    close(fdAC[0]);
    close(fdBC[1]);
    return 0;
}
```

Демонстрация работы программы

```
anetta@anetta-VirtualBox:~/Рабочий стол/kp$ ./A
Enter strings:
ret
Number of input symbols is 3
ret
Number of output symbols is 3
pert
Number of input symbols is 4
```

pert
Number of output symbols is 4

Выводы

В данном курсовом проекте я укрепила навыки работы с процессами, и узнала о возможностях взаимодействия между процессами.