Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Операционные среды и системное программирование

ОТЧЕТ
к лабораторной работе № 4
на тему

УПРАВЛЕНИЕ ПРОЦЕССАМИ И ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ

Выполнил:
студент гр. 153503
Татаринов В.В.

Проверил:
Гриценко Н.Ю.

Минск 2024

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Изучить основные особенности подсистемы управления процессами и средства взаимодействия процессов в *Unix*.

Написать программу, представляющую собой процесс, который при получении сигнала, стандартно вызывающего завершение, создает свою копию, которая продолжает выполняться с прерванного места, и лишь после этого завершается, избегая таким образом безусловного «уничтожения» не перехватываемым сигналом.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Процесс – экземпляр программы во время выполнения, независимый объект, которому выделены системные ресурсы (например, процессорное время и память). Каждый процесс выполняется в отдельном адресном пространстве: один процесс не может получить доступ к переменным и структурам данных другого. Если процесс хочет получить доступ к чужим ресурсам, необходимо использовать межпроцессное взаимодействие. Это могут быть конвейеры, файлы, каналы связи между компьютерами и многое другое.

Поток использует то же самое пространства стека, что и процесс, а множество потоков совместно используют данные своих состояний. Как правило, каждый поток может работать (читать и писать) с одной и той же областью памяти, в отличие от процессов, которые не могут просто так получить доступ к памяти другого процесса. У каждого потока есть собственные регистры и собственный стек, но другие потоки могут их использовать.

Поток – определенный способ выполнения процесса. Когда один поток изменяет ресурс процесса, это изменение сразу же становится видно другим потокам этого процесса [1].

Разделяемая память позволяет двум и более процессам использовать одну и ту же область (сегмент) оперативной памяти. Это самое эффективное средство обмена, поскольку при его использовании не происходит копирования информации, и доступ к ней производится напрямую.

Для получения доступа к сегменту разделяемой памяти используется системный вызов *shmget.* При успешном создании или, если объект с заданным ключом *key* существует, функция возвращает идентификатор сегмента. В случае ошибки функция возвращает значение -1. После создания разделяемой памяти, надо получить ее адрес. Для этого используется вызов *shmat.* Первый аргумент – идентификатор сегмента разделяемой памяти. Различные сочетания значений второго и третьего аргументов задают способ определения его адреса. Возвращаемый системным вызовом *shmat* адрес в дальнейшем используется процессом для прямого обращения к сегменту.

Изменить режим доступа к сегменту разделяемой памяти или удалить его можно при помощи системного вызова *shmctl*. При этом следует заметить, что сегмент не может быть удален до тех пор, пока не сделано обращение к процедуре *shmdt* [2].

3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ

В результате выполнения лабораторной работы был написана программа, представляющая собой процесс, который при получении сигнала, стандартно вызывающего завершение, создает свою копию, которая продолжает выполняться с прерванного места, и лишь после этого завершается.

Программа при старте запускает счетчик, который отображает каждую секунду значение в консоль. При прерывании программы происходит создание нового процесса и завершение старого. Счетчик продолжает считать с того значения, которое было в предыдущем процессе (рисунок 1).

```
Initial process. PID: 19378
Counter: 0
Counter: 1
Counter: 2
Counter: 3
^CCounter: 4
New process created. PID: 19387
Counter: 5
Counter: 6
^C
Terminating old process. PID: 19387
Counter: 7
New process created. PID: 19392
Counter: 8
Counter: 9
```

Рисунок 1 – Результат работы программы

ВЫВОДЫ

В результате выполнения лабораторной работы были изучены основные особенности подсистемы управления процессами и средства взаимодействия процессов в *Unix*.

Написана программа, представляющую собой процесс, который при получении сигнала, стандартно вызывающего завершение, создает свою копию, которая продолжает выполняться с прерванного места, и лишь после этого завершается, избегая таким образом безусловного «уничтожения» не перехватываемым сигналом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Что такое процесс [Электронный ресурс]. – Режим доступа: https://tproger.ru/problems/what-is-the-difference-between-threads-and-processes.

[2] Разделяемая память [Электронный ресурс]. – Режим доступа: https://www.sensi.org/~alec/x/x-1-7-3-3.html.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

Листинг 1 – Файл *p.c*:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include "shm.h"

int *counter;
pid_t child_pid = 0;

void handle_sigint(int sig) {
    pid_t pid;

    if (child_pid > 0) {
        printf("\nTerminating old process. PID: %d\n", child_pid);
        kill(child_pid, SIGTERM);
    }
    if ((pid = fork()) < 0) {
        perror("fork");
        exit(1);
    } else if (pid == 0) {
        printf("New process created. PID: %d\n", getpid());
    } else {
        child_pid = pid;
    }
}

int main() {
    int shmid;

    shmid = create_shared_memory(sizeof(int));
    if (shmid < 0) {
        perror("create_shared_memory");
        exit(1);
    }

    counter = attach_shared_memory(shmid);
    if (counter == (int *) -1) {
        perror("attach_shared_memory");
        exit(1);
    }

    *counter = 0;
    signal(SIGINT, handle_sigint);
    printf("Initial process. PID: %d\n", getpid());

    while (1) {
        printf("Counter: %d\n", (*counter)++);
        sleep(1);
    }
    detach_shared_memory(counter);
    remove_shared_memory(shmid);

    return 0;
}
```

Листинг 2 – Файл *shm.c*:

```
#include <stdio.h>
#include <sys/shm.h>
#include "shm.h"

int create_shared_memory(size_t size) {
    return shmget(IPC_PRIVATE, size, IPC_CREAT | 0666);
}

int *attach_shared_memory(int shmid) {
    return shmat(shmid, NULL, 0);
}

void detach_shared_memory(int *mem) {
    shmdt(mem);
}

void remove_shared_memory(int shmid) {
    shmctl(shmid, IPC_RMID, NULL);
}
```

Листинг 3 – Файл *shm.h*:

```
#ifndef SHARED_MEMORY_H
#define SHARED_MEMORY_H

#include <sys/types.h>

int create_shared_memory(size_t size);
int *attach_shared_memory(int shmid);
void detach_shared_memory(int *mem);
void remove_shared_memory(int shmid);

#endif
```

Листинг 4 – Файл *makefile*:

```
CMP = gcc

TARGET = shm

all: $(TARGET)

$(TARGET): p.o shm.o
    $(CMP) -o $@ $^

p.o: p.c shm.h
    $(CMP) -c $<

shm.o: shm.c shm.h
    $(CMP) -c $<

clean:
    rm -f *.o
```