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 3](#_Toc146836467)

[2 Теоретические сведения 4](#_Toc146836468)

[3 Полученные результаты 5](#_Toc146836469)

[Выводы 6](#_Toc146836470)

[Список использованных источников 7](#_Toc146836471)

[Приложение А (обязательное) листинг кода 8](#_Toc146836472)

# **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).

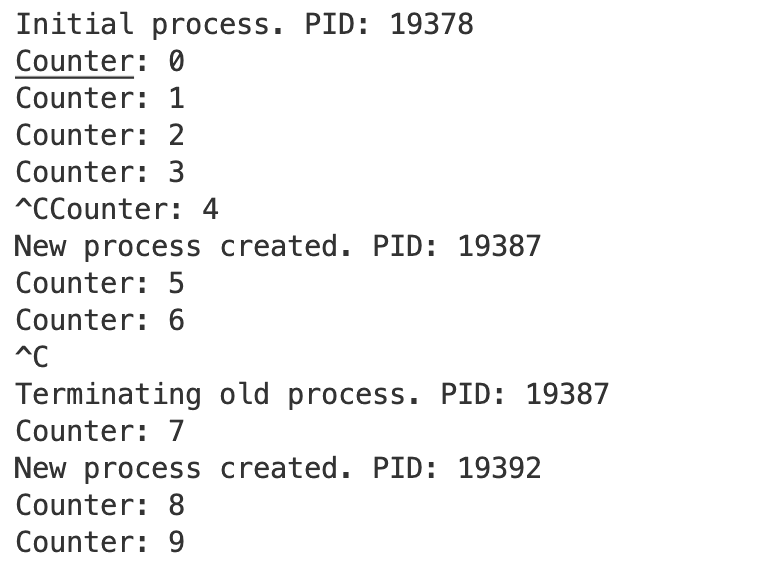


Рисунок 1 – Результат работы программы

# **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