УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе № 8

по предмету «Архитектура компьютерной техники и операционных систем»

Вариант 12

Выполнил:

Крутько А.А.

гр. 251004

Проверил:

Леванцевич В.А.

Минск 2023

Задание

1. Создать дерево процессов согласно варианта индивидуального задания.
2. Дерево процессов из таблицы 1.
3. Последовательность обмена сигналами из таблицы 2.

Процессы непрерывно обмениваются сигналами согласно табл. 2. Запись в таблице 1 вида: 1***-> (2,3,4,5)*** означает, что исходный процесс ***0*** создаёт дочерний процесс ***1,*** который, в свою очередь, создаёт дочерние процессы ***2,3,4,5.*** Запись в таблице 2 вида: 1***-> (2,3,4) SIGUSR1*** означает, что процесс 1 посылает дочерним процессам 2***,3,4*** одновременно (т.е. за один вызов kill ()) сигнал ***SIGUSR1***. После передачи ***101***–го по счету сигнала ***SIGUSR*** родительский процесс посылает сыновьям сигнал ***SIGTERM***и ожидает завершения всех сыновей, после чего завершается***.*** Сыновья, получив сигнал ***SIGTERM*** завершают работу с выводом на консоль сообщения вида:

***Pid ppid завершил работу после X-го сигналаSIGUSR1 и Y-го сигналаSIGUSR2***

***где X, Y – количество посланных за все время работы данным сыном сигналов SIGUSR1 и SIGUSR2***

Каждый процесс в процессе работы выводит на консоль информацию в следующем виде:

***Npidppid послал/получил USR1/USR2 текущее время (мксек)***

***где N-номер сына по табл. 1***

**Таблица 1. Дерево процессов**

|  |  |
| --- | --- |
| **№** | **Дерево процессов** |
| **12** | **1->(2,3,4,5,6,7) 2,3,4,5,6,7->8** |

**Таблица 2. Последовательность обмена сигналами**

|  |  |
| --- | --- |
| **№** | **Последовательность обмена сигналами** |
| **12** | **1->6 *SIGUSR1*6->7 *SIGUSR1*7->(4,5)*SIGUSR2***  **4->8 *SIGUSR1* 5->2 *SIGUSR1* 8->2*SIGUSR2* 2->1 *SIGUSR2*** |

Код программы:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <signal.h>

#include <sys/wait.h>

#include <stdbool.h>

#define PROC\_COUNT 9

static int i = 1;

static int USR1 = 0, USR2 = 0;

static bool flag = false;

pid\_t arrpid[PROC\_COUNT] = {0, 0, 0, 0, 0, 0, 0, 0, 0};

void printProcessInfo(int signum)

{

printf("%d process, pid: %d, ppid: %d, got a signal %s, time %ld\n\n", i, getpid(), getppid(), (signum == SIGUSR1) ? "SIGUSR1" : "SIGUSR2", clock());

}

void sendSignalToProcess(int processIndex, int signal, int SIGCOUNT)

{

for (int h = 1; h <= SIGCOUNT; h++)

{

kill(arrpid[processIndex], signal);

printf(">> %d process, pid: %d, ppid: %d, send signal %s to process %d, time %ld\n\n", i, getpid(), getppid(), (signal == SIGUSR1) ? "SIGUSR1" : "SIGUSR2", processIndex, clock());

usleep(100000);

}

kill(arrpid[processIndex], SIGTERM);

}

void sendSignalToGroup(int processIndex, int signal, int SIGCOUNT)

{

setpgid(arrpid[4], arrpid[4]);

setpgid(arrpid[5], arrpid[4]);

pid\_t group = getpgid(arrpid[4]);

for (int h = 1; h <= SIGCOUNT; h++)

{

killpg(group, signal);

printf(">> %d process, pid: %d, ppid: %d, send signal %s to group %d, time %ld\n\n", i, getpid(), getppid(), (signal == SIGUSR1) ? "SIGUSR1" : "SIGUSR2", group, clock());

usleep(100000);

}

killpg(group, SIGTERM);

}

void signalHandler(int signum)

{

usleep(clock());

switch (signum)

{

case SIGUSR1:

printProcessInfo(signum);

USR1++;

break;

case SIGUSR2:

printProcessInfo(signum);

USR2++;

break;

default:

flag = true;

break;

}

}

void waitForChildProcesses()

{

for (int i = 0; i < PROC\_COUNT; i++)

wait(NULL);

printf("=== %d process, pid: %d, ppid: %d, ended work after #%d signal SIGUSR1 and #%d signal SIGUSR2\n\n", i, getpid(), getppid(), USR1, USR2);

}

void createChildProcesses(FILE \*filepid, const int SIGNUM)

{

pid\_t pid = fork();

if (pid == 0)

{

arrpid[i] = getpid();

// 1 -> (2, 3, 4, 5, 6, 7)

for (int j = 2; j <= 7; j++)

{

pid = fork();

if (pid == 0)

{

i = j;

break;

}

if (pid > 0)

arrpid[j] = pid;

}

// 2, 3, 4, 5, 6, 7 -> 8

if (i != 1)

{

arrpid[i] = getpid();

pid = fork();

//childScope

if (pid == 0)

i = 8;

//parentScope

else if (pid > 0)

arrpid[8] = pid;

}

printf("pid of %d is %d, parent %d\n\n", i, getpid(), getppid());

//в самом начальном все хранится

if (i == 1)

{

arrpid[i] = getpid();

filepid = fopen("allpids.txt", "wb+");

fwrite(arrpid, sizeof(pid\_t), 8, filepid);

fclose(filepid);

}

else

usleep(10000);

filepid = fopen("allpids.txt", "rb+");

fread(&arrpid, sizeof(pid\_t), 8, filepid);

fclose(filepid);

signal(SIGUSR1, signalHandler);

signal(SIGUSR2, signalHandler);

signal(SIGTERM, signalHandler);

usleep(10000);

switch (i)

{

case 1:

sendSignalToProcess(6, SIGUSR1, SIGNUM);

break;

case 6:

sendSignalToProcess(7, SIGUSR1, SIGNUM);

break;

case 7:

sendSignalToGroup(4, SIGUSR2, SIGNUM);

break;

case 4:

sendSignalToProcess(8, SIGUSR1, SIGNUM);

break;

case 5:

sendSignalToProcess(2, SIGUSR1, SIGNUM);

break;

case 8:

sendSignalToProcess(2, SIGUSR2, SIGNUM);

break;

case 2:

sendSignalToProcess(1, SIGUSR2, SIGNUM);

break;

}

while (true)

if (flag)

break;

waitForChildProcesses();

}

else

{

printf("pid of 0 process is %d, parent %d\n\n", getpid(), getppid());

wait(NULL);

}

}

int main(int argc, char \*argv[])

{

if (argc < 2)

{

printf("use <%s> <amount\_of\_ticks>\n", argv[0]);

return 0;

}

const int SIGCOUNT = atoi(argv[1]);

FILE \*filepid;

createChildProcesses(filepid, SIGCOUNT);

return 0;

}

Блок-схема:

