УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе № 8

по предмету «Архитектура компьютерной техники и операционных систем»

Вариант 4

Выполнил:

Гузаев Е.Д.

гр. 351003

Проверил:

Леванцевич В.А.

Минск 2024

***Задание:***

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 SIGTERMи 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***

|  |  |
| --- | --- |
| **4** | **1-> (2,3,4,5) 5-> (6,7,8)** |

***Таблица 2. Последовательность обмена сигналами***

|  |  |
| --- | --- |
| **4** | ***1-> (1,2,3,4,5) SIGUSR1***  ***5-> (6,7,8) SIGUSR1***  ***8-> (1) SIGUSR1*** |

Решение:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <sys/wait.h>

#include <time.h>

#define PROC\_COUNT 9

#define SIGNAL\_COUNT 101

pid\_t arrpid[PROC\_COUNT];

volatile sig\_atomic\_t sigusr1\_count = 0;

volatile sig\_atomic\_t sigusr2\_count = 0;

void signal\_handler(int signum) {

struct timespec ts;

clock\_gettime(CLOCK\_MONOTONIC, &ts);

if (signum == SIGUSR1) {

sigusr1\_count++;

printf("PID=%d, PPID=%d received SIGUSR1 #%d at %ld.%06ld msec\n", getpid(), getppid(), sigusr1\_count, ts.tv\_sec, ts.tv\_nsec / 1000);

} else if (signum == SIGUSR2) {

sigusr2\_count++;

printf("PID=%d, PPID=%d received SIGUSR2 #%d at %ld.%06ld msec\n", getpid(), getppid(), sigusr2\_count, ts.tv\_sec, ts.tv\_nsec / 1000);

}

}

void terminate\_handler(int signum) {

printf("PID=%d, PPID=%d finished work after %d SIGUSR1 and %d SIGUSR2 signals\n", getpid(), getppid(), sigusr1\_count, sigusr2\_count);

exit(0);

}

void create\_processes() {

for (int i = 2; i <= 5; i++) {

pid\_t pid = fork();

if (pid == 0) {

printf("Process %d created: PID=%d, PPID=%d\n", i, getpid(), getppid());

arrpid[i] = getpid();

if (i == 5) {

for (int j = 6; j <= 8; j++) {

pid = fork();

if (pid == 0) {

printf("Process %d created: PID=%d, PPID=%d\n", j, getpid(), getppid());

arrpid[j] = getpid();

return;

} else {

arrpid[j] = pid;

}

}

}

return;

} else {

arrpid[i] = pid;

}

}

}

void send\_signals(int sender, int \*targets, int count) {

struct timespec ts;

clock\_gettime(CLOCK\_MONOTONIC, &ts);

for (int i = 0; i < count; i++) {

kill(arrpid[targets[i]], SIGUSR1);

printf("PID=%d sent SIGUSR1 to PID=%d at %ld.%06ld msec\n", sender, arrpid[targets[i]], ts.tv\_sec, ts.tv\_nsec / 1000);

}

}

int main() {

signal(SIGUSR1, signal\_handler);

signal(SIGUSR2, signal\_handler);

signal(SIGTERM, terminate\_handler);

arrpid[1] = getpid();

printf("Process 1 created: PID=%d, PPID=%d\n", getpid(), getppid());

create\_processes();

sleep(1); // Wait for all processes to initialize

if (getpid() == arrpid[1]) {

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

int targets[] = {2, 3, 4, 5};

send\_signals(1, targets, 4);

pause();

}

for (int i = 2; i <= 5; i++) {

kill(arrpid[i], SIGTERM);

}

} else if (getpid() == arrpid[5]) {

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

int targets[] = {6, 7, 8};

send\_signals(5, targets, 3);

}

for (int i = 6; i <= 8; i++) {

kill(arrpid[i], SIGTERM);

}

} else if (getpid() == arrpid[8]) {

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

kill(arrpid[1], SIGUSR1);

}

}

while (1) {

pause();

}

return 0;

}