Лабораторная работа № 6

ГЕНЕРАЦИЯ И ОБРАБОТКА СИГНАЛОВ

Цель работы. Освоение простейшего средства управления процессами, позволяющего процессам передавать информацию о каких-либо событиях, отрабатывать реакции на различные события и взаимодействовать друг с другом.

1. Программа sigint.cpp осуществляет ввод символов со стандартного ввода. Скомпилируйте и запустите программу и отправьте ей сигналы SIGINT (нажатием Ctrl-C) и SIGQUIT (нажатиемCtrl-\). Проанализируйте результаты.

Изображение выглядит как текст

Автоматически созданное описание

Данная программа принимает на вход строку и выводит её обратно в консоль. Если отправить сигнал SIGINT, то процесс его перехватит и выведет строку “Ahhh! SIGINT!”. Это сделано при помощи системного вызова **sigaction,** который используется для изменения действия процесса при получении соответствующего сигнала. Для сигнала SIGQUIT обработчик не установлен, поэтому он работает в штатном режиме.

2. Запустите программу signal\_catch.cpp , выполняющую вывод на консоль. Отправьте процессу сигналы SIGINT и SIGQUIT, а также SIGSTOP (нажатием Ctrl-Z) и SIGCONT (нажатием Ctrl-Q) . Проанализируйте поведение процесса и вывод на консоль, а также сравните с программой из предыдущего пункта.

Изображение выглядит как текст

Автоматически созданное описание

В отличии от программы из прошлого пункта производится обработка сигнала SIGQUIT.

Изображение выглядит как текст

Автоматически созданное описание

При остановке процесса через SIGSTOP и его продолжению по SIGCONT он перестаёт обрабатывать любые сигналы через консоль. **Протестите у себя я хз чет**

3. Скомпилируйте и запустите программу sigusr.cpp . Программа выводит на консоль значение ее PID и зацикливается, ожидая получения сигнала. Запустите второй терминал и, отправляя с него командой kill различные сигналы, в том числе и SIGUSR1 , проанализируйте реакцию на них.

Изображение выглядит как текст

Автоматически созданное описание

При отправке сигнала SIGUSR1 процесс завершается и выводится соотвествующее сообщение на консоль.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

На другие сигналы процесс реагирует в штатном режиме.

5. Составьте программу, запускающую процесс-потомок. Процесс-родитель и процесс-потомок должны генерировать (можно случайным образом) и отправлять друг другу сигналы (например, SIGUSR1, SIGUSR2). Каждый из процессов должен выводить на консоль информацию об отправленном и о полученном сигналах.

Код составленной программы предостален ниже: **зайти на cpp beatifuler и рефакторить**

|  |
| --- |
| **#include<stdio.h>**  **#include<stdlib.h>**  **#include<unistd.h>**  **#include<signal.h>**  **#include<sys/types.h>**  **main(void) {**  **void signal1(int);**  **if (signal(SIGUSR1, signal1) == SIG\_ERR) {**  **exit(1);**  **}**  **if (signal(SIGUSR2, signal1) == SIG\_ERR) {**  **exit(2);**  **}**    **int p\_pid = getpid();**  **int c\_pid = fork();**  **if (c\_pid == -1) {**  **perror("Bad fork");**  **exit(4);**  **}**  **if (c\_pid == 0) {**    **c\_pid = getpid();**  **printf("Child PID: %d\n\n", c\_pid);**    **for (int i = 1; ;i++) {**  **sleep(1);**  **switch (i % 3) {**  **case 0:**  **kill(c\_pid, SIGUSR1);**  **break;**  **case 1:**  **kill(p\_pid, SIGUSR1);**  **break;**  **default:**  **kill(p\_pid, SIGUSR2);**  **}**  **}**  **} else {**  **printf("Parent PID: %d\n", p\_pid);**    **for (int i = 1;; i++) {**  **sleep(1);**  **switch (i % 3) {**  **case 0:**  **kill(c\_pid, SIGUSR1);**  **break;**  **case 1:**  **kill(c\_pid, SIGUSR2);**  **break;**  **default:**  **kill(p\_pid, SIGUSR2);**  **}**  **}**  **}**  **}**  **void signal1(int the\_sig) {**  **signal(the\_sig,signal1);**  **printf("Signal %d received on %d\n\n", the\_sig,getpid());**  **}** |

Итог выполнения программы:

Изображение выглядит как текст

Автоматически созданное описание

6. Для организации обработчиков сигналов предпочтительно использовать системный вызов sigaction() и соответствующую структуру данных. Обеспечьте корректное завершение процессов.

Для более лучше организации обработчиков использовался системный вызов sigaction():

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <errno.h>  #include <signal.h>  int main(void) {  void signal\_p(int);  void signal\_c(int);  struct sigaction sa;  sa.sa\_flags = 0;  sigemptyset( & sa.sa\_mask);  int p\_pid = getpid();  int c\_pid = fork();  if (c\_pid == -1) {  perror("Bad fork");  exit(4);  }  if (c\_pid == 0) {  c\_pid = getpid();  printf("Child PID: %d\n\n", c\_pid);  sa.sa\_handler = signal\_c;  if (sigaction(SIGUSR1, &sa, NULL) == -1) {  printf("Signal not received");  perror("SIGQUIT");  exit(1);  }  if (sigaction(SIGUSR2, &sa, NULL) == -1) {  printf("Signal not received");  perror("SIGQUIT");  exit(1);  }  for (int i = 1;; i++) {  sleep(1);  switch (i % 3) {  case 0:  kill(c\_pid, SIGUSR1);  break;  case 1:  kill(p\_pid, SIGUSR1);  break;  default:  kill(p\_pid, SIGUSR2);  }  }  } else {  printf("Parent PID: %d\n", p\_pid);  sa.sa\_handler = signal\_p;  if (sigaction(SIGUSR1, &sa, NULL) == -1) {  printf("Signal not received");  perror("SIGQUIT");  exit(1);  }  if (sigaction(SIGUSR2, &sa, NULL) == -1) {  printf("Signal not received");  perror("SIGQUIT");  exit(1);  }  for (int i = 1;; i++) {  sleep(1);  switch (i % 3) {  case 0:  kill(c\_pid, SIGUSR1);  break;  case 1:  kill(c\_pid, SIGUSR2);  break;  default:  kill(p\_pid, SIGUSR2);  }  }  }  }  void signal\_p(int the\_sig) {  signal(the\_sig, signal\_p);  printf("Parent received %d\n\n", the\_sig);  if (the\_sig == SIGQUIT) {  exit(1);  }  }  void signal\_c(int the\_sig) {  signal(the\_sig, signal\_c);  printf("Child received %d\n\n", the\_sig);  if (the\_sig == SIGQUIT) {  exit(1);  }  } |

Изображение выглядит как текст

Автоматически созданное описание

7. Модифицируйте программу занятия 3 (файлы pipe\_server.cpp , pipe\_client.cpp и pipe\_local.h), сделав ее более стабильной вработе. В числе недостатков, которые желательно устранить, можно указать: если клиентский процесс завершается по получению сигнала SIGINT (Ctrl+C), то private FIFO не удаляется из системы (исправляется посредством организации перехвата сигнала с выполнением необходимых действий); клиентский процесс при его инициализации может обрушиться, если сервер окажется недоступен (исправляется путем попытки запуска сервера из клиента, если сервер не активен).

Пусть кто-то сделает, у меня нет идей что с этим чудищем делать….