МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО"

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №2**

з дисципліни “Системне програмування”

тема “Управління процесами, відокремлення процесів”

Виконав студент 3 курсу

групи КП-92

Сусленко Матвій Олегович

**Тема**: Єдиний державний реєстр осіб, які вчинили корупційні правопорушення

Київ 2021

Завдання

**Частина 1: Запуск, завершення та параметри процесу.**

Скласти програму, яка:

1. Отримує та друкує інформацію про параметри свого процесу за допомогою системних

викликів getpid(), getgid(), getsid() тощо.

2. Виконує розгалудження процесу за допомогою системного виклику fork().

3. Для процесу-батька та процесу-нащадка окремо роздруковує їхні ідентифікатори у циклі.

4. Виконати очікування процесом-батьком завершення нащадка.

5. Повідомляє про завершення процесів — батька та нащадка.

6. Пояснити отримані результати

|  |
| --- |
| #include <unistd.h>  #include <stdio.h>  #include <sys/wait.h>  #include <errno.h>  #include <string.h>  #include <stdlib.h>  #define NUMBER\_OF\_ITERATIONS 100  void throwError(char\* message) {  fprintf(stderr, "%s: %s\n", message, strerror(errno));  exit(1);  }  void printInfo(char\* key) {  pid\_t currentPID = getpid();  gid\_t currentGID = getgid();  pid\_t currentSID = getsid(currentPID);  if (currentSID == -1) {  throwError("Error while trying to get SID of process");  }  printf("[%s] PID: %i\n", key, currentPID);  printf("[%s] GID: %i\n", key, currentGID);  printf("[%s] SID: %i\n", key, currentSID);  printf("\n");  }  void printProcessesInfo(char\* key) {  for (int i = 0; i < NUMBER\_OF\_ITERATIONS; i++) {  printf("%i: ", i);  printInfo(key);  }  }  int main() {  printInfo("Initial");  pid\_t childProcessPID = fork();  if(childProcessPID < 0)  {  throwError("Fork failed");  }  if (childProcessPID == 0) {  printProcessesInfo("Child");  printf("Child process ended...\n");  return 0;  } else {  int resultInfo;  printProcessesInfo("Parent");  wait(&resultInfo);  printf("\nParent process ended...\nStatus: %i\n", resultInfo);  return 0;  }  return 0;  } |

**Частина 2: Демонізація процесу.**

Скласти програму, яка:

1. Відкриває на запис текстовий файл логу та робить у нього запис про старт програми.

2. Виконує демонізацію свого функціоналу за таким алгоритмом:

* Виконує fork()
* Для процесу-батька робить запис у лог про породження нащадка та закриває себе викликом exit().
* Для поцесу-нащадка:
* Робить setsid()
* Змінює поточний каталог на “/”
* Закриває усі відкриті батьком дескриптори
* Відкриває “/dev/null” на запис для трьох стандартних потоків

3. Відкриває лог та робить у нього запис про параметри демонізованого процесу.

4. Виконує витримку часу у нескінченному циклі.

5. Пояснити отримані результати

|  |
| --- |
| #include <stdio.h>  #include <fcntl.h>  #include <string.h>  #include <errno.h>  #include <stdlib.h>  #include <unistd.h>  #include <time.h>  #define REQUIRED\_ARG\_COUNT 2  #define WRITE\_FILE\_CREATE\_RULES 0644  void writeToFile(int fileDescriptor, char\* str) {  write(fileDescriptor, str, strlen(str));  }  void throwError(int fileDescriptor, char\* message) {  writeToFile(fileDescriptor, message);  exit(1);  }  void closeAllParentDescriptors() {  for (int i = 0; i < 255; i++) {  close(i);  }  }  int openFileToWrite(char\* fileName) {  int fileDescriptor = open(fileName, O\_WRONLY | O\_CREAT | O\_TRUNC, WRITE\_FILE\_CREATE\_RULES);  if (fileDescriptor == -1) {  fprintf(stderr, "%s: %s\n", "Write file can't be opened!\n", strerror(errno));  exit(1);  }  return fileDescriptor;  }  int main(int argc, char\* argv[]) {  if (argc != REQUIRED\_ARG\_COUNT) {  fprintf(stderr, "%s: %s\n", "Invalid arguments count\n", strerror(errno));  exit(1);  }  char \* writeFileName = argv[1];  int writeFileDescriptor = openFileToWrite(writeFileName);  writeToFile(writeFileDescriptor, "Parent Process started\n");  pid\_t childPid = fork();  if (childPid < 0) {  throwError(writeFileDescriptor, "Fork failed");  }  if (childPid == 0) {  pid\_t newSid = setsid();  if (newSid == -1) {  throwError(writeFileDescriptor, "Error while calling setsid\n");  }  if (chdir("/") == -1) {  throwError(writeFileDescriptor, "Error while calling chdir\n");  }  closeAllParentDescriptors();  open("/dev/null", O\_RDWR);  dup(0);  dup(0);  int newFileDescriptor = openFileToWrite(writeFileName);  char buffer[100];  int bufferLength = sprintf(buffer, "PID: %d, GID: %d, SID: %d\n", getpid(), getgid(), newSid);  write(newFileDescriptor, buffer, bufferLength);  while(1) {  time\_t cur = time(NULL);  char \* stime = ctime(&cur);  bufferLength = sprintf(buffer, "PID: %d, Current time is: %s", getpid(), stime);  write(newFileDescriptor, buffer, bufferLength);  sleep(5);  }  close(newFileDescriptor);  return 0;  } else {  writeToFile(writeFileDescriptor, "Parent Process exit\n");  exit(EXIT\_SUCCESS);  }  return 0;  } |