МИНОБРНАУКИ РОССИИ

ФГБОУ ВО «ИЖЕВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ М. Т. КАЛАШНИКОВА»

Кафедра «Программное обеспечение»

Отчет

по лабораторной работе № 1

по дисциплине

«Периферийные устройства и системное программирование»

|  |  |
| --- | --- |
| Выполнил:  студент гр. Б07-191-1 | Р. И. Мусин |
| Принял:  Доцент, к.т.н. | М.А. Аль Аккад |

Ижевск 2018

Постановка задачи

Необходимо реализовать программы, которые показывают работу Unix-системы, например, управление процессами, стеками, файловой системой и т.п.

Текст программы

myfork.c

#include **<stdio.h>**#include **<unistd.h>  
  
int** main() {  
 **int** pid;  
 printf(**"I am parent with PID %d\n"**, getpid());  
 pid = fork();  
 **if** (pid != 0) {  
 printf(**"I am parent with PID %d\n"**, getpid());  
 printf(**"My child PID %d\n"**, pid);  
 } **else** {  
 printf(**"I am child PID %d\n"**, getpid());  
 }  
 printf(**"PID %d"**, getpid());  
}

zombie.c

#include **<stdio.h>**#include **<unistd.h>**#include **<stdlib.h>  
  
int** main() {  
 **int** pid = fork();  
 **if** (pid) {  
 **while** (1);  
 } **else** {  
 printf(**"parent exit"**);  
 exit(42);  
 }  
}

background.c

#include **<unistd.h>**#include **<stdio.h>  
  
int** main(**int** argc, **char** \*argv[]) {  
 **if** (fork() == 0) {  
 execvp(argv[1], &argv[1]);  
 fprintf(**stderr**, **"Couldn't execute argv %c"**, argv[1]);  
 }  
}

pipe.c

#include **<unistd.h>**#include **<stdio.h>**#define **READ** 0  
#define **WRITE** 1  
  
**int** fd[2];  
  
**void** write(**char** \*argv[]) {  
 close(fd[**READ**]);  
 dup2(fd[**WRITE**], 1);  
 close(fd[**WRITE**]);  
 execlp(argv[1], argv[1], **NULL**);  
 perror(**"connect"**);  
}  
  
**void** read(**char** \*argv[]) {  
 close(fd[**WRITE**]);  
 dup2(fd[**READ**], 0);  
 close(fd[**READ**]);  
 execvp(argv[2], &argv[2]);  
 perror(**"connect"**);  
}  
  
**int** main(**int** argc, **char** \*argv[]) {  
 pipe(fd);  
 **if** (fork() != 0) {  
 write(argv);  
 } **else** {  
 read(argv);  
 }  
 **return** 0;  
}

alarm.c

#include **<stdio.h>**#include **<unistd.h>  
  
int** main() {  
 alarm(3);  
 printf(**"Looking for..."**);  
 **while**(1);  
 printf(**"This line will never be executed"**);  
}

alarmHandler.c

#include **<stdio.h>**#include **<unistd.h>**#include **<wait.h>  
  
bool** alarmFlag = **false**;  
  
**void** alarmHandler(**int** unused) {  
 printf(**"alarm signal was recieved\n"**);  
 alarmFlag = **true**;  
}  
  
**int** main() {  
 signal(**SIGALRM**, alarmHandler);  
 alarm(3);  
 printf(**"Looping...\n"**);  
 **while** (!alarmFlag) {  
 pause();  
 printf(**"Loop ends due to alarm signal\n"**);  
 }  
}

2stepWrite.c

#include **<stdio.h>**#include **<fcntl.h>**#include **<unistd.h>  
  
int** main() {  
 **int** fd = open(**"1.txt"**, **O\_WRONLY**);  
 write(fd, **"hi there\n"**, 9);  
 lseek(fd, 0, **SEEK\_SET**);  
 fcntl(fd, **F\_SETFL**, **O\_WRONLY** | **O\_APPEND**);  
 write(fd, **" Abc\n"**, 6);  
 close(fd);  
 **return** 0;  
}

diodcontroller.c

#include **<stdio.h>**#include **<fcntl.h>**#include **<unistd.h>**#include **<sys/ioctl.h>**#include **<linux/kd.h>  
  
int** diodController() {  
 **int** td = open(**"/dev/tty0"**, **O\_NOCTTY**);  
 **if** (td == -1) {  
 perror(**"open"**);  
 **return** -1;  
 }  
 **int** state;  
 **if** (ioctl(td, **KDGKBLED**, &state) == -1) {  
 perror(**"ioctl"**);  
 **return** -1;  
 }  
 state ^= **K\_NUMLOCK**;  
 **if** (ioctl(td, **KDGKBLED**, &state)) {  
 perror(**"ioctl set"**);  
 close(td);  
 **return** -1;  
 }  
 close(td);  
 **return** 0;  
}  
  
**int** main() {  
 **while** (**true**) {  
 **char** t = getchar();  
 **switch** (t) {  
 **case ' '**:  
 diodController();  
 **break**;  
 **default**:  
 **break**;  
 }  
 }  
}

reader.c

#include **<stdio.h>**#include **<fcntl.h>**#include **<unistd.h>**#include **<sys/stat.h>  
  
bool** readline(**int** fd, **char** \*str) {  
 ssize\_t n;  
 **do** {  
 n = read(fd, str, 1);  
 } **while** (n > 0 && \*str++ != 0);  
 **return** (n > 0);  
}  
  
**int** main() {  
 **int** fd;  
 **char** str[100];  
 mkfifo(**"apipe"**, 0660);  
 fd = open(**"apipe"**, **O\_RDONLY**);  
 **while** (readline(fd, str)) {  
 printf(**"%s\n"**, str);  
 }  
 close(fd);  
}

writer.c

#include **<stdio.h>**#include **<fcntl.h>**#include **<string.h>**#include **<unistd.h>  
  
int** main() {  
 **int** fd;  
 **char** \*message = **new char**[10];  
 sprintf(message, **"Hello form pid %d"**, getpid());  
 size\_t messagelen = strlen(message) + 1;  
 **do** {  
 fd = open(**"apipe"**, **O\_WRONLY**);  
 **if** (fd == -1) {  
 sleep(1);  
 }  
 } **while** (fd == -1);  
 **for** (**int** i = 1; i <= 3; i++) {  
 write(fd, message, messagelen);  
 sleep(3);  
 }  
 close(fd);  
}

showerror.c

#include **<fcntl.h>**#include **<errno.h>**#include **<stdio.h>  
  
int** main() {  
 **int** fd;  
 fd = open(**"nonexist.txt"**, **O\_RDONLY**);  
 **if** (fd == -1) {  
 printf(**"error number %d\n"**, **errno**);  
 perror(**"File doesn't exist"**);  
 }  
 fd = open(**"/"**, **O\_WRONLY**);  
 **if** (fd == -1) {  
 printf(**"error = %d\n"**, **errno**);  
 perror(**"permisson denied"**);  
 }  
}

pulse.c

#include **<signal.h>**#include **<unistd.h>**#include **<cstdio>  
  
int** main() {  
 **int** Pid1, Pid2;  
 Pid1 = fork();  
 **if** (Pid1 == 0) {  
 **while** (**true**) {  
 printf(**"Pid1 alive\n"**);  
 sleep(2);  
 }  
 }  
 Pid2 = fork();  
 **if** (Pid2 == 0) {  
 **while** (**true**) {  
 printf(**"Pid2 alive\n"**);  
 sleep(2);  
 }  
 }  
  
 kill(Pid1, **SIGSTOP**); *//Приостановить Pid1* sleep(3);  
 kill(Pid1, **SIGCONT**); *//Возобновить Pid1* sleep(3);  
 kill(Pid1, **SIGINT**); *// Убить pid1* kill(Pid2, **SIGINT**); *// Убить pid2* **return** 0;  
}

stack.c

#include **<stdio.h>**#define **STACK\_SIZE** 10  
  
**struct** Stack {  
 **int** items[**STACK\_SIZE**];  
 **int** size = 0;  
  
 **int** pop() {  
 **return** items[--size];  
 }  
  
 **void** push(**int** item) {  
 items[size++] = item;  
 }  
};  
  
**int** main() {  
 Stack st;  
 **for** (**int** i = 0; i < 10; i++) {  
 **int** item;  
 scanf(**"%d"**, &item);  
 st.push(item);  
 }  
 **for** (**int** i = 0; i < 10; i++)  
 printf(**"%d"**, st.pop());  
 **return** 0;  
}

queue.c

#include **<stdio.h>**#define **QUEUE\_SIZE** 50  
  
**struct** Queue {  
 **int** items[**QUEUE\_SIZE**];  
 **int** head, size;  
  
 **bool** isEmpty() {  
 **return** head == size;  
 }  
  
 **int** push(**int** item) {  
 items[size++] = item;  
 }  
  
 **int** pop() {  
 **return** items[head++];  
 }  
};  
  
**int** main() {  
 Queue queue;  
  
 **while** (1) {  
 **int** item;  
 scanf(**"%d"**, &item);  
 **if** (item == 0)  
 **break**;  
 queue.push(item);  
 }  
 **while** (!queue.isEmpty())  
 printf(**"%d "**, queue.pop());  
 printf(**"\n"**);  
  
 **return** 0;  
}

peterson.c

#include **<stdio.h>**#include **<unistd.h>  
  
int** turn;  
**int** interested[2];  
  
**void** enter\_region(**int** process) {  
 **int** other;  
 other = 1 - process;  
 interested[process] = 1;  
 turn = process;  
 **while** (turn == process && interested[other] == 1);  
 printf(**"enter process %d \n"**, process);  
}  
  
**void** leave\_region(**int** process) {  
 interested[process] = 0;  
 printf(**"leave process %d \n"**, process);  
}  
  
**int** main() {  
 **int** ppid = fork();  
 **if** (ppid > 0) {  
 enter\_region(0);  
 } **else** {  
 enter\_region(1);  
 }  
 **if** (ppid > 0) {  
 leave\_region(0);  
 } **else** {  
 leave\_region(1);  
 }  
}

Вывод

Мы рассмотрели основные команды Unix-системы, которые могут пригодиться нам при повседневном использовании системы.