System Software Practicals

Assignment 1

Krunal Rank
U18CO081

1. To study the basics of system call and system library.

The given reference material has been studied.

2. Programs for the simulation of following system calls :
Fork, exec, getpid, exit, wait, close, stat, opendir, readdir,chmod,chown

```
#include <bits/stdc++.h>// Library for common data structures and algos
#include <stdio.h>      // Standard IO
#include <unistd.h>
#include <sys/types.h>  // For common syscalls
#include <sys/wait.h>   // For wait
#include <fcntl.h>      // For File handling
#include <sys/stat.h>   // For attributes of file
#include <dirent.h>     // For directory handling
#include <libgen.h>     //

using namespace std;

void fork_example(){
   cout<<"FORK EXAMPLE"<<endl;
   /*  Clone the calling process, creating an exact copy
       -1 : error
       0 : child process
       positive val: parent process*/
   pid_t id = fork();
   int cstatus;

   if(id==-1){
       cerr<<errno<<endl;
       exit(0);
   }else{
       cout<<id<<endl;
       if(id==0) {
           cout<<"child process executed!"<<endl;
           /* Exits the calling process with a status code*/
           exit(1);
       }
```

```
        else wait(&cstatus); // Waits the calling process until child process has
exited
    }
}

void exec_example(){
    /*  Executes commands and arguments
        @param
        command : command that you need to execute
        args    : arguments  */
    cout<<"EXEC EXAMPLE"<<endl;
    int cstatus;
    pid_t child = fork();
    if(child==-1){
        exit(0);
    }else{
        if(child==0){
            char* args[] = {"ls","-a",NULL};
            execvp(args[0],args);
            exit(1);
        }else{
            wait(&cstatus);
        }
    }
}

void getpid_example(){
    /*  Returns pid of calling process */
    cout<<"PID EXAMPLE"<<endl;
    cout<<getpid()<<endl;
}

void file_example(){
    cout<<"FILE EXAMPLE"<<endl;
    // Creates a file in disk with required flags RDWR = Read/Write
    int fd =creat("temp.txt",O_RDWR);

    struct stat sfile;
    // Gets attributes for a file
    stat("temp.txt", &sfile);
    cout<<"ST MODE: "<<sfile.st_mode<<endl;

    if(fd==-1){
        cout<<"Failed to create a file!"<<endl;
```

```
        exit(0);
    }
    cout<<"Created file with fd: "<<fd<<endl;

    // Writes in file
    write(fd,"Hello World! This is a string",strlen("Hello World! This is a string"));
    cout<<"Written in file!"<<endl;

    char* y = (char* )calloc(100,sizeof(char));

    // Read a file
    int fd1 = open("temp.txt",O_RDONLY);
    int end = read(fd1,y,90);
    y[end] = '\0';
    printf("Read from file: %s\n",y);

    // Closes a file
    close(fd);
    close(fd1);

    cout<<"Closed file!"<<endl;
}

void dir_example(){
    cout<<"DIRECTORY EXAMPLE"<<endl;
    DIR *dir;
    struct dirent * dp;

    // Opens the directory
    if((dir=opendir("."))==NULL){
        cout<<"Error in opening current folder!"<<endl;
        exit(0);
    }
    cout<<"Reading Directory"<<endl;

    // Reads the directory
    while((dp=readdir(dir))!=NULL){
        printf("%s\n",dp->d_name);
    }

    // Closes directory
    closedir(dir);
    cout<<"Closed Directory"<<endl;
```

```
}

void chmod_example(){
   cout<<"CHMOD EXAMPLE"<<endl;
   char mode[] = "0777";
   char buf[100] = "temp.txt";
   int i;
   i = strtol(mode, 0, 8);
   if (chmod (buf,i) < 0)
   {
       cout<<"Error in chmod of temp.txt"<<" "<<strerror(errno)<<endl;
       exit(1);
   }
   cout<<"CHMOD successful!"<<endl;
}

void chown_example(){
   cout<<"CHOWN EXAMPLE"<<endl;

   // Gets UID
   uid_t uid = getuid();
   cout<<"User ID: "<<uid<<endl;

   if(chown("temp.txt",uid,-1)==-1){
       cout<<"Error in chown syscall! Error: "<<strerror(errno)<<endl;
       exit(0);
   }
   cout<<"CHOWN Successful!"<<endl;
}

int main(){

   fork_example();
   exec_example();
   getpid_example();
   file_example();
   dir_example();
   chmod_example();
   chown_example();

   return 0;

}
```

```
krhero@hellblazer:/mnt/0FB812900FB81290/BTech/Assignments/3rd_Year/SS/Assignment1$ ./a.out
FORK EXAMPLE
43919
0
child process executed!
EXEC EXAMPLE
.  ..  2.cpp  a.out  tempt.txt  temp.txt
PID EXAMPLE
43918
FILE EXAMPLE
ST MODE: 33279
Created file with fd: 6
Written in file!
Read from file: Hello World! This is a string
Closed file!
DIRECTORY EXAMPLE
Reading Directory
.
..
2.cpp
a.out
temp.txt
tempt.txt
Closed Directory
CHMOD EXAMPLE
CHMOD successful!
CHOWN EXAMPLE
User ID: 1000
CHOWN Successful!
```

3. Program for the simulation of following System calls:
read, write, open, close, poll, lseek, mmap, munmap, brk, rt_sigaction, rt_sigprocmask, pread64,pwrite64, readv, writev, alarm,getittimer, setittimer, getpid, socket, connect, accept, sendto, recvfrom, sendmsg, recvmsg, shutdown, bind, listen, getsocketname, exit, kill, pipe, pause.

Some of the syscalls in this question have been covered in the previous question.

```
#include <bits/stdc++.h> // Library for common data structures and algos
#include <stdio.h>       // Standard IO
#include <unistd.h>
#include <sys/types.h>  // For common syscalls
#include <sys/wait.h>   // For wait
#include <fcntl.h>      // For File handling
#include <sys/stat.h>   // For attributes of file
#include <dirent.h>     // For directory handling
#include <libgen.h>     // General purpose library
#include <sys/poll.h>   // For polling
#include <sys/mman.h>   // For mmap munmap
#include <signal.h>     // For signals
#include <sys/time.h>   // For Timer
#include <netdb.h>      // For socket programming
#include <netinet/in.h> // For socket programming

using namespace std;

void read_example()
{
   cout << "READ EXAMPLE" << endl;
   char *x = (char *)calloc(100, sizeof(char));
   read(STDIN_FILENO, x, 10);
   cout << "Read from file: ";
   write(STDOUT_FILENO, x, 10);
   cout << endl;
}

void poll_example()
{
   cout << "POLL EXAMPLE" << endl;

   struct pollfd fds[3];
   int ret;

   fds[0].fd = STDIN_FILENO;
   fds[0].events = POLLIN;
```

```
    ret = poll(fds, 3, 5 * 1000);
    if (ret == -1)
    {
        cout << "Polling timeout!" << endl;
        exit(0);
    }

    if (!ret)
    {
        printf("%d seconds elapsed.\n", 5);
    }

    if (fds[0].revents & POLLIN)
    {
        printf("Read Event Successful\n");
    }
}

void lseek_example()
{
    cout << "LSEEK EXAMPLE" << endl;

    int f_write = open("temp.txt", O_RDONLY);
    int f_read = open("tempt.txt", O_WRONLY);

    int count = 0;
    char arr[100];
    int n;
    n = 5;
    while (read(f_write, arr, 1))
    {
        if (count < n)
        {
            lseek(f_write, n, SEEK_CUR);
            write(f_read, arr, 1);
            count = n;
        }
        else
        {
            count = (2 * n);
            lseek(f_write, count, SEEK_CUR);
            write(f_read, arr, 1);
        }
    }
```

```
    close(f_write);
    close(f_read);
    cout << "LSEEK Successful!" << endl;
}

void map_example()
{
    cout << "MAP EXAMPLE" << endl;
    char *addr;
    int fd;
    struct stat sb;
    off_t offset, pa_offset;
    size_t length;
    ssize_t s;

    fd = open("temp.txt", O_RDONLY);
    offset = 2;
    pa_offset = offset & ~(sysconf(_SC_PAGE_SIZE) - 1);
    if (offset >= sb.st_size)
    {
        fprintf(stderr, "offset is past end of file\n");
        exit(EXIT_FAILURE);
    }

    length = 5;

    addr = (char *)mmap(NULL, length + offset - pa_offset, PROT_READ,
                        MAP_PRIVATE, fd, pa_offset);
    if (addr == MAP_FAILED)
        cout << "MAP Error! Error: " << strerror(errno) << endl;

    s = write(STDOUT_FILENO, addr + offset - pa_offset, length);
    cout << endl;

    munmap(addr, length + offset - pa_offset);
    close(fd);
}

void p_example()
{
    cout << "PWRITE EXAMPLE" << endl;
    pwrite64(open("tempt.txt", O_WRONLY), "Hello World", strlen("Hello World"), 0);

    cout << "PREAD EXAMPLE" << endl;
```

```
   char *x = (char *)calloc(100, sizeof(char));
   pread64(open("temp.txt", O_RDONLY), x, 10, 0);
   printf("Read Text: %s\n", x);
}

void alarm_example()
{
   cout << "ALARM EXAMPLE" << endl;

   // Setting alarm
   alarm(2);
   for (int i = 1; i < 3; i++)
   {

       printf("%d : Inside main function\n", i);
       sleep(1);
   }
}

void sig_handler(int signum)
{

   printf("ALARM Successful\n");
}

unsigned int x = 0;
void interrupt(int signum)
{
   printf("timer %d!\n", ++x);
}

void timer_example()
{
   cout << "TIMER EXAMPLE" << endl;
   struct sigaction sa;
   struct itimerval timer;

   memset(&sa, 0, sizeof(sa));
   sa.sa_handler = &interrupt;

   // Sigaction handler
   sigaction(SIGPROF, &sa, NULL);

   timer.it_value.tv_sec = 0;
```

```
    timer.it_value.tv_usec = 5000;
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 5000;

    // Set timer example
    setitimer(ITIMER_PROF, &timer, NULL);
}

void kill_example()
{
    cout << "KILL EXAMPLE" << endl;

    cout << "Process will get killed now!" << endl;
    kill(getpid(), 0);
}

void pause_example()
{
    cout << "PAUSE EXAMPLE" << endl;
    int ret = 0;
    ret = pause();
    printf("Pause returned with %d\n", ret);
}

void pipe_example()
{
    cout << "PIPE EXAMPLE" << endl;
    int pipefds[2];
    char *pin;
    char buffer[5];

    if (pipe(pipefds) == -1)
    {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();

    if (pid == 0)
    {
        pin = "4821\0";
        close(pipefds[0]);
        write(pipefds[1], pin, 5);
```

```
        printf("Generating data in child and sending to parent...\n");
        sleep(2); // intentional delay
        exit(EXIT_SUCCESS);
    }

    if (pid > 0)
    {
        wait(NULL);
        close(pipefds[1]);
        read(pipefds[0], buffer, 5);
        close(pipefds[0]);

        printf("Parent received data '%s'\n", buffer);
    }
}


int main()
{
    signal(SIGALRM, sig_handler);
    // read_example();
    // poll_example();
    // lseek_example();
    // map_example();
    // p_example();
    // alarm_example();
    // timer_example();
    // kill_example();
    // pause_example();
    // pipe_example();

    return 0;
}
```

```
krhero@hellblazer:/mnt/0FB812900FB81290/BTech/Assignments/3rd_Year/SS/Assignment1$ ./a.out
READ EXAMPLE
0123456789
0123456789Read from file:
```

```
krhero@hellblazer:/mnt/0FB812900FB81290/BTech/Assignments/3rd_Year/SS/Assignment1$ ./a.out
POLL EXAMPLE
a
Read Event Successful
```

```
krhero@hellblazer:/mnt/0FB812900FB81290/BTech/Assignments/3rd_Year/SS/Assignment1$ ./a.out
LSEEK EXAMPLE
LSEEK Successful!
```

```
krhero@hellblazer:/mnt/0FB812900FB81290/BTech/Assignments/3rd_Year/SS/Assignment1$ ./a.out
MAP EXAMPLE
llo W
```

```
krhero@hellblazer:/mnt/0FB812900FB81290/BTech/Assignments/3rd_Year/SS/Assignment1$ ./a.out
PWRITE EXAMPLE
PREAD EXAMPLE
Read Text: Hello Worl
```

```
ALARM EXAMPLE
1 : Inside main function
2 : Inside main function
ALARM Successful
```

```
TIMER EXAMPLE
KILL EXAMPLE
Process will get killed now!
```

```
krhero@hellblazer:/mnt/0FB812900FB81290/BTech/Assignments/3rd_Year/SS/Assignment1$ ./a.out
PAUSE EXAMPLE

^C
```

```
krhero@hellblazer:/mnt/0FB812900FB81290/BTech/Assignments/3rd_Year/SS/Assignment1$ ./a.out
PIPE EXAMPLE
Generating data in child and sending to parent...
Parent received data '4821'
```

Server:

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#define PORT 8080
int main(int argc, char const *argv[])
{
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};
    char *hello = "Server Handshake Message";

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
```

```
    {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                                                &opt, sizeof(opt)))
    {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons( PORT );

    if (bind(server_fd, (struct sockaddr *)&address,
                                 sizeof(address))<0)
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    if (listen(server_fd, 3) < 0)
    {
        perror("listen");
        exit(EXIT_FAILURE);
    }
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
                        (socklen_t*)&addrlen))<0)
    {
        perror("accept");
        exit(EXIT_FAILURE);
    }
    valread = read( new_socket , buffer, 1024);
    printf("%s\n",buffer );
    send(new_socket , hello , strlen(hello) , 0 );
    printf("Handshake Complete\n");
    return 0;
}
```

Client:

```
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
```

```c
#include <unistd.h>
#include <string.h>
#define PORT 8080

int main(int argc, char const *argv[])
{
  int sock = 0, valread;
  struct sockaddr_in serv_addr;
  char *hello = "Client Handshake Message";
  char buffer[1024] = {0};
  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  {
     printf("\n Socket creation error \n");
     return -1;
  }

  serv_addr.sin_family = AF_INET;
  serv_addr.sin_port = htons(PORT);

  if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0)
  {
     printf("\nInvalid address/ Address not supported \n");
     return -1;
  }

  if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
  {
     printf("\nConnection Failed \n");
     return -1;
  }
  send(sock, hello, strlen(hello), 0);
  printf("Handshake Request Sent\n");
  valread = read(sock, buffer, 1024);
  printf("%s\n", buffer);
  printf("Handshake Complete\n");
  return 0;
}
```

```
krhero@hellblazer:/mnt/0FB812900FB81290/BTech/Assignments/3rd_Year/SS/Assignment1$ ./server.out
Server Started! Listening for Connections...
Client Handshake Message
Handshake Complete
```

```
krhero@hellblazer:/mnt/0FB812900FB81290/BTech/Assignments/3rd_Year/SS/Assignment1$ ./client.out
Handshake Request Sent
Server Handshake Message
Handshake Complete
```