NIT SRINAGAR

National Institute of Technology, Srinagar

NAME: Burhaan Rasheed Zargar

ENROLLMENT: 2022BITE012

DEPARTMENT: ITE

COURSE: OS Lab (ITL-252)

TABLE OF CONTENTS

Implement the process manipulation system calls like fork(), exec().

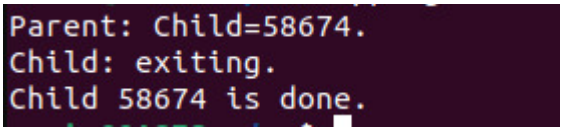
1.Fork System Call

Code:

program1.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
int main(){
    int pid=fork();
    if(pid>0)
    {
        printf("Parent: Child=%d.\n", pid);
        pid=wait(NULL);
        printf("Child %d is done. \n", pid);
    }
    else if(pid==0)
    {
        printf("Child: exiting.\n");
        exit(0);
    }
    else
    {
        printf("Fork error.\n");
    }
}

```

Output:



2.Exec System Call

Code:

program.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
int main(){
    pid_t i;
    char *args[2];
    args[0]="/bin/ls";
    args[1]=NULL;
    i=fork();
    if(i==0){
        execv(args[0],args);
    }
    else{
        wait(NULL);
        printf("Parent process %d \n", getpid());
        printf("Child process %d \n", i);
    }
    return 0;
}

```

Output:

```
dup     file.c     open     os1.c     program     program.c  signal
dup.c   message    open.c   pipes     program1    shared     signal.c
file    message.c  os1      pipes.c   program1.c  shared.c
Parent process 59647
Child process 59648
```

Implement the file manipulation system calls like open(), close().

1.Open and Close System Call

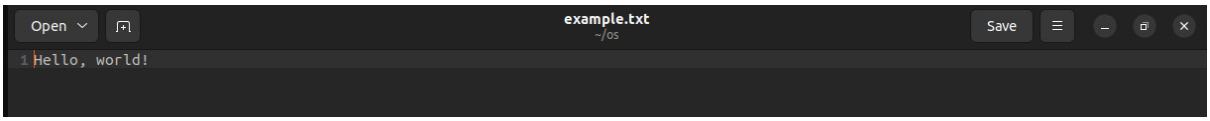
Code:

open.c

```c
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>

int main(void)
{
    int fd;
    char buf[12] = "hello world";

    fd = open("new.txt", O_RDWR | O_CREAT, 0666);
    if (fd == -1) {
        perror("Error opening file");
        return 1; // Return an error code
    }

    ssize_t bytes_written = write(fd, buf, strlen(buf));
    if (bytes_written == -1) {
        perror("Error writing to file");
        close(fd);
        return 1; // Return an error code
    }

    printf("Successfully wrote %zd bytes to the file.\n", bytes_written);

    if (close(fd) == -1) {
        perror("Error closing file");
        return 1; // Return an error code
    }

    return 0;
}

```

Output:

```
Successfully wrote 11 bytes to the file.
```



Implement the system calls for manipulating the file descriptors.

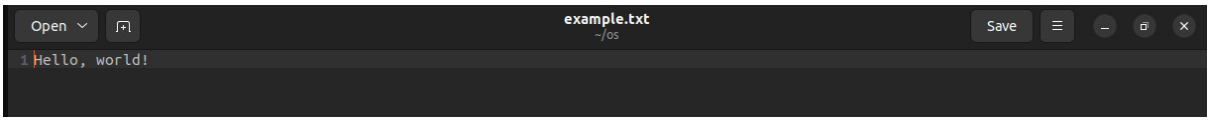
1.Dup System Call

Code:

dup.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int main() {
    int fd1, fd2;

    // Open a file
    fd1 = open("example.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd1 < 0) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    // Duplicate the file descriptor
    fd2 = dup(fd1);
    if (fd2 < 0) {
        perror("dup");
        exit(EXIT_FAILURE);
    }

    // Write to the original file descriptor
    if (write(fd1, "Hello, ", 7) < 0) {
        perror("write fd1");
        exit(EXIT_FAILURE);
    }

    // Write to the duplicated file descriptor
    if (write(fd2, "world!\n", 7) < 0) {
        perror("write fd2");
        exit(EXIT_FAILURE);
    }

    // Close file descriptors
    close(fd1);
    close(fd2);

    return 0;
}


```

Output:



Implement the system calls for Inter-Process communication.

1.Pipes

Code:

pipes.c

```c
#include <unistd.h>
#include <stdio.h>

int main() {
    int fd[2];
    char buffer[20];

    pipe(fd);

    if (fork() == 0) { // Child process
        close(fd[0]); // Close read end
        write(fd[1], "Hello World!", 13); // Write data to pipe
        close(fd[1]); // Close write end
    } else { // Parent process
        close(fd[1]); // Close write end
        read(fd[0], buffer, 20); // Read data from pipe
        printf("%s\n", buffer);
        close(fd[0]); // Close read end
    }

    return 0;
}

```

Output:

2.Message Passing

Code:

message.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

struct msgbuf {
    long mtype;
    char mtext[100];
};

int main() {
    int msqid;
    key_t key;
    struct msgbuf message;

    // Create message queue
    key = ftok("/tmp", 'a');
    msqid = msgget(key, IPC_CREAT | 0666);

    // Send message to message queue
    message.mtype = 1;
    sprintf(message.mtext, "Hello, world!");
    msgsnd(msqid, &message, sizeof(message), 0);

    // Receive message from message queue
    msgrcv(msqid, &message, sizeof(message), 1, 0);
    printf("Received message: %s\n", message.mtext);

    // Remove message queue
    msgctl(msqid, IPC_RMID, NULL);

    return 0;
}

```

Output:

```
Received message: Hello, world!
```

2.Shared Memory

Code:

shared.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>

int main() {
    int shmid;
    int *shared_data;
    pid_t pid;
    const int SHM_SIZE = sizeof(int);

    // Create shared memory segment
    shmid = shmget(IPC_PRIVATE, SHM_SIZE, IPC_CREAT | 0666);

    // Attach to shared memory segment
    shared_data = shmat(shmid, NULL, 0);

    // Fork a child process
    pid = fork();
    if (pid < 0) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) { // Child process
        // Child writes to shared memory
        *shared_data = 42;
        printf("Child process wrote %d to shared memory.\n", *shared_data);
    } else { // Parent process
        // Wait for the child to complete
        wait(NULL);

        // Parent reads from shared memory
        printf("Parent process read %d from shared memory.\n", *shared_data);
    }

    // Detach from shared memory segment
    shmdt(shared_data);

    // Remove shared memory segment
    shmctl(shmid, IPC_RMID, NULL);

    return 0;
}


```

Output:

```
Child process wrote 42 to shared memory.
Parent process read 42 from shared memory.
```

Implement a multi-threaded program that is doing the matrix multiplication using multiple threads.

Code:

matrice.c

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define N 3  // Define the size of the matrices

int A[N][N] = {{1, 0, 1}, {0, 1, 1}, {1, 0, 1}};
int B[N][N] = {{0, 2, 2}, {2, 0, 2}, {0, 2, 2}};
int C[N][N] = {{0}};  // Resultant matrix initialized to zero

// Structure to hold arguments for the thread function
typedef struct {
    int row;
} ThreadArgs;

// Function to be executed by each thread
void* addRows(void* arg) {
    ThreadArgs* args = (ThreadArgs*) arg;
    int row = args->row;
    for (int i = 0; i < N; i++) {
        C[row][i] = A[row][i] + B[row][i];
    }
    return NULL;
}

int main() {
    pthread_t threads[N];  // Declare thread identifiers
    ThreadArgs args[N];    // Argument structures for each thread

    // Initialize arguments for each thread and create the threads
    for (int i = 0; i < N; i++) {
        args[i].row = i;
        if (pthread_create(&threads[i], NULL, addRows, &args[i]) != 0) {
            perror("Error creating thread");
            return 1;
        }
    }

    // Wait for all threads to complete
    for (int i = 0; i < N; i++) {
        if (pthread_join(threads[i], NULL) != 0) {
            perror("Error joining thread");
            return 1;
        }
    }

    // Print the resulting matrix
    printf("Resulting Matrix C:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d ", C[i][j]);
        }
        printf("\n");
    }

    return 0;
```

Output:

Implementing the new signal handlers for the standard signals.

Code:

signal.c

```c
#define _POSIX_C_SOURCE 1
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>

void handler_func(int sig){
    printf("Edited handler here!\n");
    exit(1);
}

int signal_def(int sig, void (*handler)(int)){
    struct sigaction act;
    act.sa_handler=handler;
    sigemptyset(&act.sa_mask);
    act.sa_flags=0;
    return sigaction(sig,&act,NULL);
}

int main(){
    if(signal_def(SIGINT,handler_func)==-1){
        fprintf(stderr,"Can't map handler!\n");
        exit(2);
    }
    for(long i=0;i<1000000;i++)
        printf("%li\n", i);
    return 0;
}

```

Output:

```
529561
529562
529563
529564
529565
529566
529567
529568
529569
529570
529571
529572
529573
529574
529575
529576
529577
529578
^C529692
Edited handler here!
```

Understand the use of free and pmap utilities in Linux.

1.Free

```
oasis@OASIS:~$ free -m
               total        used        free      shared  buff/cache   available
Mem:            5916        2811         387         125        2718        2691
Swap:           2047           0        2047
```

2.Pmap

```
oasis@OASIS:~/os$ pmap -x 3
3:   [rcu_gp]
Address           Kbytes     RSS   Dirty Mode  Mapping
---------------- ------- ------- -------
total kB               0       0       0
```

ISO file creation for LINUX kernel.

```
wget https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.8.9.tar.xz
tar xf linux-6.8.9.tar.xz
cd linux-6.8.9/
make defconfig
apt-get update
apt-get install flex
apt-get install bison
apt-get install libelf-dev
apt-get install libssl-dev
make defconfig
make -j $(nproc)
find -name "bzImage"
wget https://busybox.net/downloads/busybox-1.36.1.tar.bz2
tar xf busybox-1.36.1.tar.bz2
cd busybox-1.36.1
make -j $(nproc)
file busybox
make install
cd _install
mkdir dev proc sys
touch init
gedit init
--Enter the following in init file:
        #!/bin/sh
        mount -t devtmpfs none /dev
        mount -t proc none /proc
        mount -t sysfs none /sys
        echo "Welcome to my Linux!"
        exec /bin/sh
chmod +x init
find . -print0 | cpio --null -ov --format=newc | gzip -9 > ../initramfs.cpio.gz
qemu-system-x86_64 -kernel linux-6.8.9/arch/x86_64/boot/bzImage -initrd busybox-
1.36.1/initramfs.cpio.gz
mkdir iso
cd iso
mkdir boot
cd boot
mkdir grub
--Copy bzImage and initramfs.cpio.gz into boot folder
cd grub
touch grub.cfg
gedit grub.cfg
--Enter the following code:
        set default=0
        set timeout=10
        menuentry 'myos' --class os {
        insmod gzio
        insmod part_msdos
        linux /boot/bzImage
        initrd /boot/initramfs.cpio.gz
        }
sudo apt-get update
sudo apt-get install xorriso
```

sudo apt-get update
sudo apt-get install mtools
grub-mkrescue -o myos.iso iso/
qemu-system-x86_64 -cdrom myos.iso -m 1024