**Week2**

[s2022103042@centos8-linux Wed May 15 05:58 PM os]$ cd wk1

[s2022103042@centos8-linux Wed May 15 05:58 PM wk1]$ ls

**a1** **commands** h1.txt h2.txt h3.txt

[s2022103042@centos8-linux Wed May 15 05:58 PM wk1]$ cd ..

[s2022103042@centos8-linux Wed May 15 05:58 PM os]$ cd wk2

[s2022103042@centos8-linux Wed May 15 05:58 PM wk2]$ ls

**q1.sh** **q2.sh** **q3.sh** **q4.sh** **q5.sh** **q6.sh** **q7.sh** **s2.sh** **s3.sh** **s4.sh** **s5.sh** **s6.sh** **s7.sh**

[s2022103042@centos8-linux Wed May 15 05:58 PM wk2]$ cat q1.ah

cat: q1.ah: No such file or directory

[s2022103042@centos8-linux Wed May 15 06:00 PM wk2]$ cat q1.sh

#!/bin/bash

read -p "enter a value of n:" n

power=$((2\*\*n))

for((i=1;i<=power;i\*=2))

do

echo $i;

done

[s2022103042@centos8-linux Wed May 15 06:00 PM wk2]$ cat q2.sh

#!/bin/bash

read -p"enter num1:" num1

read -p"enter num2:" num2

sum=$((num1+num2))

echo "sum is $sum"

[s2022103042@centos8-linux Wed May 15 06:00 PM wk2]$ cat q3.sh

#!/bin/bash

if [ $# -ne 3 ]; then

echo "usage:$0 <num1><operator><num2>"

exit 1

fi

num1=$1

operator=$2

num2=$3

if ! [[ $num1 =~ ^[0-9]+$ ]] || ! [[ $num2 =~ ^[0-9]+$ ]]; then

echo "error: provide numeric value for num1 and num2"

exit 1

fi

case $operator in

+)

result=$((num1+num2));;

-)

result=$((num1-num2));;

\\*)

result=$((num1\*num2));;

/)

if [ $num2 -eq 0 ];then

echo "error: division by zero is not possible"

exit 1

fi

result=$((num1/num2));;

\*)

echo "invalid operation"

exit1;;

esac

echo "result:$result"

[s2022103042@centos8-linux Wed May 15 06:00 PM wk2]$ cat q4.sh

#!/bin/bash

if [ $# -lt 2 ]; then

echo "usage: $0 <num1><num2>[<num3>...]"

exit 1

fi

max=$1

for x in "$@"; do

if [[ $x =~ ^[0-9]+$ ]]; then

if [ $x -gt $max ]; then

max=$x

fi

else

echo "error: $x is not numeric"

fi

done

echo "the largest value is $max"

[s2022103042@centos8-linux Wed May 15 06:00 PM wk2]$ cat q5.sh

#!/bin/bash

echo "enter a number"

read n

reverse=""

for (( i=${#n}-1; i>=0; i-- )); do

reverse="$reverse${n:$i:1}"

done

echo "original number:" $n

echo "reversed number:" $reverse

[s2022103042@centos8-linux Wed May 15 06:00 PM wk2]$ cat q6.sh

#!/bin/bash

read -p"enter the n:" n

a=0

b=1

while [ $a -le $n ]

do

echo $a

temp=$((a+b))

a=$b

b=$temp

done

[s2022103042@centos8-linux Wed May 15 06:00 PM wk2]$ cat q7.sh

#!/bin/bash

echo "enter string done for completed"

declare -a strings

while true; do

read str

if [ "$str" == "done" ]; then

break

fi

strings+=("$str")

done

for string in "${strings[@]}"; do

len=${#string}

echo "string: $string, length: $len"

done

[s2022103042@centos8-linux Wed May 15 06:00 PM wk2]$ cat q8.sh

cat: q8.sh: No such file or directory

[s2022103042@centos8-linux Wed May 15 06:00 PM wk2]$ cat s2.sh

#!/bin/bash

for((i=0;i<8;i++))

do

for((j=0;j<8;j++))

do

if (( (i+j) % 2==0))

then

echo -ne "\033[47m \033[0m"

else

echo -ne "\033[40m \033[0m"

fi

done

echo

done

[s2022103042@centos8-linux Wed May 15 06:01 PM wk2]$ cat s3.sh

#!/bin/bash

hour=$(date +%H)

if (( hour >=1 && hour < 12));

then

echo "good morning"

elif ((hour >12 && hour <18));

then

echo "good afternoon"

elif ((hour >=18 && hour <20));

then

echo "good evening"

else

echo "good night"

fi

[s2022103042@centos8-linux Wed May 15 06:01 PM wk2]$ cat s4.sh

#!/bin/bash

read -p"enter a string" str

echo $str | tr ’[:upper:]’ ’[:lower:]’

echo $str | tr ’[:lower:]’ ’[:upper:]’

[s2022103042@centos8-linux Wed May 15 06:01 PM wk2]$ cat s5.sh

#!/bin/bash

read -p"enter n" n

for((i=0;i<n;i++))

do

for((j=0;j<i;j++))

do

echo -ne "\* "

done

echo

done

[s2022103042@centos8-linux Wed May 15 06:01 PM wk2]$ cat s6.sh

#!/bin/bash

chars='A-Za-z0-9!@#$%^&\*(){}[]/\.;'

pass=$(head /dev/urandom | tr -dc "$chars" |head -c 8)

echo "$pass"

[s2022103042@centos8-linux Wed May 15 06:01 PM wk2]$ cat s7.sh

#!/bin/bash

read -p"enter month:" mon

read -p"enter year:" year

cal $mon $year

**Week 3**

[s2022103042@centos8-linux Wed May 15 06:01 PM wk2]$ cd ..

[s2022103042@centos8-linux Wed May 15 06:01 PM os]$ cd wk3

[s2022103042@centos8-linux Wed May 15 06:01 PM wk3]$ ls

employee.txt fruit.txt output.txt sample.txt

[s2022103042@centos8-linux Wed May 15 06:01 PM wk3]$ cat employee.txt

ajay manager account 45000

sunil clerk account 25000

varun manager sales 50000

amit manager account 47000

tarun peon sales 15000

deepak clerk sales 23000

sunil peon sales 13000

satvik director purchase 80000

[s2022103042@centos8-linux Wed May 15 06:02 PM wk3]$ cat fruit.txt

fruit cost quantity

apple 120 1kg

mango 70 1kg

grapes 50 500g

banana 45 12p

watermelon 80 1p

papaya 20 1p

bcd

[s2022103042@centos8-linux Wed May 15 06:02 PM wk3]$ cat output.txt

ajay manager account 45000

amit manager account 47000

satvik director purchase 80000

[s2022103042@centos8-linux Wed May 15 06:02 PM wk3]$ cat sample.txt

unix is great os. unix was developed in Bell labs.

learn operating system.

Unix linux which one you choose.

uNix is easy to ulearn. unix is a multiuser os. Learn unix. unix is a powerful.

**Week 4**

[s2022103042@centos8-linux Wed May 15 06:02 PM wk3]$ cd ..

[s2022103042@centos8-linux Wed May 15 06:02 PM os]$ cd wk4

[s2022103042@centos8-linux Wed May 15 06:02 PM wk4]$ ls

**cam.txt** **ex10** **ex2** **ex4** **ex6** **ex8** **file.txt** p11.c p2.c p4.c p6.c p8.c samp.txt sp1.c sp2.c

**ex1** **ex11** **ex3** **ex5** **ex7** **ex9** p10.c p1.c p3.c p5.c p7.c p9.c sam.txt **sp2**

[s2022103042@centos8-linux Wed May 15 06:02 PM wk4]$ cat p1.c

#include<sys/stat.h>

#include<fcntl.h>

#include<stdio.h>

#include<string.h>

#include<unistd.h>

int main()

{

open("file.txt",O\_CREAT,S\_IRWXU);

}

[s2022103042@centos8-linux Wed May 15 06:02 PM wk4]$ cat p2.c

#include<sys/stat.h>

#include<fcntl.h>

#include<stdio.h>

#include<string.h>

#include<unistd.h>

int main(){

int fd;

char x[15]="HELLO";

fd=open("file.txt",O\_RDWR);

write(fd,x,strlen(x));

close(fd);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:02 PM wk4]$ cat p3.c

#include<sys/stat.h>

#include<fcntl.h>

#include<string.h>

#include<stdio.h>

#include<unistd.h>

int main()

{

int fd;

char y[15];

fd=open("file.txt",O\_RDWR);

read(fd,y,15);

printf("%s\n",y);

close(fd);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:02 PM wk4]$ cat p4.c

#include<sys/stat.h>

#include<fcntl.h>

#include<string.h>

#include<stdio.h>

#include<unistd.h>

int main(){

int fd;

char rbuf[15];

fd=open("file.txt",O\_RDWR);

lseek(fd,-1,SEEK\_END);

read(fd,rbuf,1);

printf("%s\n",rbuf);

close(fd);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:03 PM wk4]$ cat p5.c

#include<sys/types.h>

#include<fcntl.h>

#include<string.h>

#include<stdio.h>

#include<unistd.h>

void main(){

int old\_fd,new\_fd;

old\_fd=open("file.txt",O\_RDWR);

printf("file description is %d\n",old\_fd);

new\_fd=dup(old\_fd);

printf("new file description is %d\n",new\_fd);

int close(int old\_fd);

int close(int new\_fd);

}

[s2022103042@centos8-linux Wed May 15 06:03 PM wk4]$ cat p6.c

#include<sys/stat.h>

#include<fcntl.h>

#include<string.h>

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{

char str1[5],str2[10];

int n,fd,fd1,n1;

fd=open("file.txt",O\_APPEND|O\_RDWR);

fd1=dup(fd);

n=read(fd,str1,5);

write(1,str1,n);

printf("\n");

write(1,"enter a string",15);

n1=read(0,str2,n1);

write(fd1,str2,n1);

int close(int fd1);

int close(int fd);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:03 PM wk4]$ cat p7.c

#include<stdio.h>

#include<dirent.h>

#include<string.h>

int main()

{

DIR \*dir;

struct dirent \*a;

dir=opendir(".");

while((a=readdir(dir))!=NULL){

int n=strlen(a->d\_name);

if(a->d\_name[n-1]=='t' && a->d\_name[n-2]=='x' && a->d\_name[n-3]=='t' && a->d\_name[n-4]=='.')

printf(".TXTFILE %s\n",a->d\_name);

}

closedir(dir);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:03 PM wk4]$ cat p8.c

#include<stdio.h>

#include<dirent.h>

int main()

{

DIR \*dir;

struct dirent \*a;

char str[15];

printf("enter the path:");

scanf("%s",str);

dir=opendir(str);

while((a=readdir(dir))!=NULL){

printf("FILE: %s\n",a->d\_name);

}

closedir(dir);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:03 PM wk4]$ cat p9.c

#include<dirent.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<sys/types.h>

#include<stdio.h>

#include<stdlib.h>

int main()

{

struct dirent \*\* namelist;

int n=scandir(".",&namelist,NULL,alphasort);

while(n--){

printf("%s\n",namelist[n]->d\_name);

free(namelist[n]);

}

free(namelist);

exit(EXIT\_SUCCESS);

}

[s2022103042@centos8-linux Wed May 15 06:03 PM wk4]$ cat p10.c

#include<stdio.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<sys/types.h>

int main(){

int fd,fd1;

fd=open("file.txt",O\_RDWR|O\_APPEND);

printf("old file description: %d\n",fd);

fd1=dup2(fd,1);

printf("new file description: %d\n",fd1);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:03 PM wk4]$ cat p11.c

#include<dirent.h>

#include<string.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<string.h>

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{

char str1[50],str2[50];

int n,fd,fd1,n1;

fd=open("file.txt",O\_RDWR);

DIR \*dir;

struct dirent \*a;

dir=opendir(".");

while((a=readdir(dir))!=NULL){

int n=strlen(a->d\_name);

if(a->d\_name[n-1]=='t' && a->d\_name[n-2]=='x' && a->d\_name[n-3]=='t' && a->d\_name[n-4]=='.' && a->d\_name[n-5]=='m' && a->d\_name[n-6]=='a' &

& a->d\_name[n-7]=='c'){

write(1,"do you need to over write:(0/1)",25);

n1=read(0,str2,25);

if(str2==0){

fd1=open("cam.txt",O\_WRONLY);

}

}

int cnt=1;

}

fd1=open("cam.txt",O\_CREAT,S\_IRWXU);

read(fd,str1,50);

write(fd1,str1,50);

printf("\n");

int close(int fd1);

int close(int fd);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:03 PM wk4]$ cat p12.c

cat: p12.c: No such file or directory

[s2022103042@centos8-linux Wed May 15 06:03 PM wk4]$ ls

**cam.txt** **ex10** **ex2** **ex4** **ex6** **ex8** **file.txt** p11.c p2.c p4.c p6.c p8.c samp.txt sp1.c sp2.c

**ex1** **ex11** **ex3** **ex5** **ex7** **ex9** p10.c p1.c p3.c p5.c p7.c p9.c sam.txt **sp2**

[s2022103042@centos8-linux Wed May 15 06:03 PM wk4]$ cat sp1.c

nclude<sys/stat.h>

#include<fcntl.h>

#include<stdio.h>

#include<string.h>

#include<unistd.h>

int main(){

int fd;

char str[50];

fd=open("file.txt",O\_RDWR);

read(fd,str,100);

int fd1,n;

fd1=open("d.txt",O\_RDWR);

printf("\n%d",fd1);

if(fd1==4){

//open("fil.txt",O\_CREAT,S\_IRWXU);

//

// printf("\nDo you want to overwrite?\n");

//

// scanf("%d",&n);

//

// if(n==1){

//

//

write(fd1,str,strlen(str));

//

//

}

//

//

else{

//

//

printf("\nCan't overwrite\n");

//

//

}

//

//

}

//

//

if(fd1==-1){

//

//

open("d.txt",O\_CREAT,S\_IRWXU);

//

//

int fd2=open("d.txt"

,O\_RDWR);

//

//

writ

e(fd2,str,strlen(str));

//

//

}

//

//

//

//

}

[s2022103042@centos8-linux Wed May 15 06:03 PM wk4]$ cat sp2.c

#include<stdio.h>

#include<time.h>

int main()

{

time\_t now;

struct tm \* current\_time;

time(&now);

current\_time=localtime(&now);

current\_time->tm\_mday+=2;

mktime(current\_time);

printf("Two day:%02d|%02d|%d%02d:%02d:%o2d \n" ,current\_time->tm\_mon+1,current\_time->tm\_mda

y,current\_time->tm\_year+1900,current\_time->tm\_hour,current\_time->tm\_min,current\_time->tm\_sec);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:03 PM wk4]$ cat sam.txt

cat: sam.txt: Permission denied

[s2022103042@centos8-linux Wed May 15 06:04 PM wk4]$ cat samp.txt

cat: samp.txt: Permission denied

[s2022103042@centos8-linux Wed May 15 06:04 PM wk4]$ ls

**cam.txt** **ex10** **ex2** **ex4** **ex6** **ex8** **file.txt** p11.c p2.c p4.c p6.c p8.c samp.txt sp1.c sp2.c

**ex1** **ex11** **ex3** **ex5** **ex7** **ex9** p10.c p1.c p3.c p5.c p7.c p9.c sam.txt **sp2**

**Week 5**

[s2022103042@centos8-linux Wed May 15 06:04 PM wk4]$ cd ..

[s2022103042@centos8-linux Wed May 15 06:04 PM os]$ cd wk5

[s2022103042@centos8-linux Wed May 15 06:04 PM wk5]$ ls

**ex1** **ex2** **ex3** **ex4** **ex5** **ex6** **ex7** **ex8** p1.c p2.c p3.c p4.c p5.c p6.c p7.c p8.c

[s2022103042@centos8-linux Wed May 15 06:04 PM wk5]$ cat p1.c

#include <stdio.h>

#include <dirent.h>

int main() {

DIR \*dir = opendir(".");

if (dir == NULL) {

perror("Error opening directory");

return 1;

}

closedir(dir);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:04 PM wk5]$ cat p2.c

#include <stdio.h>

#include <dirent.h>

int main() {

DIR \*dir = opendir(".");

if (dir == NULL)

{

perror("Error opening directory");

return 1;

}

struct dirent entry, \*result;

while (readdir\_r(dir, &entry, &result) == 0 && result != NULL)

{

printf("%s\n", entry.d\_name);

}

closedir(dir);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:04 PM wk5]$ cat p3.c

#include <stdio.h>

#include <sys/stat.h>

int main(){

struct stat info;

if (lstat(".", &info) == -1) {

perror("Error getting file status");

return 1;

}

printf("File type: ");

if (S\_ISREG(info.st\_mode))

printf("Regular file\n");

else if (S\_ISDIR(info.st\_mode))

printf("Directory\n");

else if (S\_ISLNK(info.st\_mode))

printf("Symbolic link\n");

else

printf("Unknown\n");

printf("Size: %ld bytes\n", info.st\_size);

printf("Last modified time: %ld\n", info.st\_mtime);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:04 PM wk5]$ cat p4.c

#include <stdio.h>

#include <dirent.h>

int main() {

DIR \*dir = opendir(".");

if (dir == NULL) {

perror("Error opening directory");

return 1;

}

struct dirent \*entry;

while ((entry = readdir(dir)) != NULL) {

printf("%s\n", entry->d\_name);

}

rewinddir(dir);

printf("\nAfter rewind:\n");

while ((entry = readdir(dir)) != NULL) {

printf("%s\n", entry->d\_name);

}

closedir(dir);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:04 PM wk5]$ ct p5.c

bash: ct: command not found...

Similar command is: 'tc'

[s2022103042@centos8-linux Wed May 15 06:04 PM wk5]$ cat p5.c

#include <stdio.h>

#include <stdlib.h>

#include <ftw.h>

int display\_info(const char \*fpath, const struct stat \*sb, int typeflag) {

printf("%s\n", fpath); return

0; // Continue traversal

}

int main() {

if (ftw(".", display\_info, 20) == -1) {

perror("Error walking directory");

return 1;

}

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:05 PM wk5]$ cat p6.c

#include <stdio.h>

#include <sys/stat.h>

#include<time.h>

int main() {

struct stat info;

if (stat(".", &info) == -1) {

perror("Error getting file status");

return 1;

}

printf("File type: ");

if (S\_ISREG(info.st\_mode))

printf("Regular file\n");

else if (S\_ISDIR(info.st\_mode))

printf("Directory\n");

else if (S\_ISLNK(info.st\_mode))

printf("Symbolic link\n");

else

printf("Unknown\n");

printf("Size: %ld bytes\n", info.st\_size);

printf("Permissions: %o\n", info.st\_mode & 0777);

printf("Owner ID: %d\n", info.st\_uid);

printf("Group ID: %d\n", info.st\_gid);

printf("Last access time: %s", ctime(&info.st\_atime));

printf("Last modification time: %s", ctime(&info.st\_mtime));

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:05 PM wk5]$ cat p7.c

#include <stdio.h>

#include <dirent.h>

#include <string.h>

int main() {

DIR \*dir;

struct dirent \*entry;

long int pos;

dir = opendir(".");

if (dir == NULL) {

perror("Unable to open directory");

return 1;

}

pos = telldir(dir);

printf("Current position in directory: %ld\n", pos);

rewinddir(dir);

pos = telldir(dir);

printf("Position after rewind: %ld\n", pos);

seekdir(dir, 2);

int skipped = 0;

while ((entry = readdir(dir)) != NULL) {

if (strcmp(entry->d\_name, ".") == 0 || strcmp(entry->d\_name, "..") == 0) {

skipped++;

continue;

}

if (skipped > 0) {

printf("%d entry in directory: %s\n", skipped + 1, entry->d\_name);

break;

}

}

closedir(dir);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:05 PM wk5]$ cat p8.c

#include <stdio.h>

#include <dirent.h>

#include <unistd.h>

#include <errno.h>

int main() {

DIR \*dir;

struct dirent \*entry;

dir = opendir(".");

if (dir == NULL) {

perror("Unable to open directory");

return 1;

}

int dir\_fd = dirfd(dir);

if (dir\_fd == -1) {

perror("Failed to obtain file descriptor");

closedir(dir);

return 1;

}

printf("Directory listing:\n");

while ((entry = readdir(dir)) != NULL) {

printf("%s\n", entry->d\_name);

}

closedir(dir);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:05 PM wk5]$ ls

**ex1** **ex2** **ex3** **ex4** **ex5** **ex6** **ex7** **ex8** p1.c p2.c p3.c p4.c p5.c p6.c p7.c p8.c

[s2022103042@centos8-linux Wed May 15 06:05 PM wk5]$

**Week 6**

[s2022103042@centos8-linux Wed May 15 06:07 PM os]$ cd wk6

[s2022103042@centos8-linux Wed May 15 06:07 PM wk6]$ ls

**ex1** **ex11** **ex2** **ex4** **ex6** **ex8** p10.c p1.c p3.c p5.c p7.c p9.c s1.c s2.c

**ex10** **ex12** **ex3** **ex5** **ex7** **ex9** p11.c p2.c p4.c p6.c p8.c **s1** **s2** **spot**

[s2022103042@centos8-linux Wed May 15 06:07 PM wk6]$ cat p1.c

#include<stdio.h>

#include<unistd.h>

int main(int argc,char \*arg[]){

pid\_t pid=fork();

printf("current pid= %d\ncurrent ppid= %d\n",getpid(),getppid());

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:08 PM wk6]$ cat p2.c

#include<unistd.h>

#include<stdio.h>

int main()

{

int i;

printf("parent pid= %d\n",getppid());

for(i=0; i<3; i++) fork();

printf("current pid= %d\t current parent pid= %d\n", getpid(), getppid());

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:08 PM wk6]$ cat p3.c

#include <stdio.h>

#include <unistd.h>

int main() {

execl("/bin/ls", "ls", "-1",NULL);

printf("This line will not be executed\n");

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:08 PM wk6]$ cat p4.c

#include<stdio.h>

int main(int argc, char\* argv[])

{

printf("%s %s\n",argv[0],argv[1]);

}

[s2022103042@centos8-linux Wed May 15 06:08 PM wk6]$ cat p5.c

#include<stdio.h>

#include<unistd.h>

#include <sys/wait.h>

int main()

{

pid\_t p1 = fork();

pid\_t p2 = fork();

if(!(p1 == 0 && p2 != 0)) wait(NULL);

if(p1 == 0 && p2 != 0) goto there; if(p1 != 0 && p2 == 0) wait(NULL);

if(!(p1 == 0 && p2 == 0)) wait(NULL); if(p1 == 0 && p2 == 0) goto there;

if(!(p1 != 0 && p2 != 0)) wait(NULL);

there:

if(p1 != 0 && p2 != 0)

printf("Parent pid= %d\n", getpid());

else

printf("Child pid= %d with parent pid= %d\n", getpid(), getppid());

}

[s2022103042@centos8-linux Wed May 15 06:08 PM wk6]$ cat p6.c

#define \_GNU\_SOURCE

#include <stdio.h>

#include <unistd.h>

#include <sched.h>

#include <stdlib.h>

#include <sys/wait.h>

#define STACK\_SIZE 65536 // 64 KB stack size

int child\_func(void \*arg) {

printf("Child process\n");

return 0;

}

int main() {

char \*stack = malloc(STACK\_SIZE);

if (stack == NULL) {

perror("malloc");

return 1;

}

pid\_t pid = clone(child\_func, stack + STACK\_SIZE, CLONE\_VM | SIGCHLD, NULL);

if (pid == -1) {

perror("clone");

return 1;

}

printf("Parent process\n");

waitpid(pid, NULL, 0);

free(stack);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:08 PM wk6]$ cat p7.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <unistd.h>

int main() {

int pid = fork();

if (pid == 0) {

printf("Child process\n");

sleep(2);

exit(0);

} else if (pid > 0) {

printf("Parent process\n");

int status;

wait(&status);

printf("Child process exited with status: %d\n", WEXITSTATUS(status));

} else {

fprintf(stderr, "Fork failed\n");

return 1;

}

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:08 PM wk6]$ cat p8.c

#include <stdio.h>

#include <stdlib.h>

int main() {

printf("Before exit\n");

exit(0);

printf("After exit\n"); // This line will not be executed

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:08 PM wk6]$ cat p9.c

#include <stdio.h>

#include <pthread.h>

void \*thread\_func(void \*arg) {

printf("Thread running\n");

pthread\_exit(NULL);

}

int main() {

pthread\_t tid;

pthread\_create(&tid, NULL, thread\_func, NULL);

pthread\_join(tid, NULL);

printf("Main thread exiting\n");

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:08 PM wk6]$ cat p10.c

#include <stdio.h>

#include <pthread.h>

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

int counter = 0;

void \*thread\_func(void \*arg) {

pthread\_mutex\_lock(&mutex);

counter++;

printf("Thread incrementing counter: %d\n", counter);

pthread\_mutex\_unlock(&mutex);

pthread\_exit(NULL);

}

int main() {

pthread\_t tid[5];

for (int i = 0; i < 5; i++) {

pthread\_create(&tid[i], NULL, thread\_func, NULL);

}

for (int i = 0; i < 5; i++) {

pthread\_join(tid[i], NULL);

}

printf("Final counter value: %d\n", counter);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:08 PM wk6]$ cat p11.c

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

sem\_t semaphore;

int counter = 0;

void \*thread\_func(void \*arg) {

sem\_wait(&semaphore);

counter++;

printf("Thread incrementing counter: %d\n", counter);

sem\_post(&semaphore);

pthread\_exit(NULL);

}

int main() {

sem\_init(&semaphore, 0, 1);

pthread\_t tid[5];

for (int i = 0; i < 5; i++) {

pthread\_create(&tid[i], NULL, thread\_func, NULL);

}

for (int i = 0; i < 5; i++) {

pthread\_join(tid[i], NULL);

}

printf("Final counter value: %d\n", counter);

sem\_destroy(&semaphore);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:08 PM wk6]$ cat p12.c

cat: p12.c: No such file or directory

[s2022103042@centos8-linux Wed May 15 06:08 PM wk6]$ ls

**ex1** **ex11** **ex2** **ex4** **ex6** **ex8** p10.c p1.c p3.c p5.c p7.c p9.c s1.c s2.c

**ex10** **ex12** **ex3** **ex5** **ex7** **ex9** p11.c p2.c p4.c p6.c p8.c **s1** **s2** **spot**

[s2022103042@centos8-linux Wed May 15 06:08 PM wk6]$ cat s1.c

#include<stdio.h>

#include<unistd.h>

void main(){

int i;

for(i=1;i<=4;i++){

if(fork()==0){

printf(" %d child process created\n",i);

break;

}

}

}

[s2022103042@centos8-linux Wed May 15 06:08 PM wk6]$ cat s2.c

#include<stdio.h>

#include<unistd.h>

void createbinarytree(int depth,int currentdepth){

if(currentdepth>=depth){

return ;

}

if(fork()==0){

printf("\nCHILD PID : %d , PARENT PID : %d\n",getpid(),getppid());

createbinarytree(depth,currentdepth+1);

}

}

int main(){

int depth;

printf("ENTER THE NUMBER OF NODES IN THE BINARY TREE : ");

scanf("%d",&depth);

printf("PARENT PID : %d\n",getpid());

createbinarytree(depth,0);

return 0;

}

**Week 7**

[s2022103042@centos8-linux Wed May 15 05:53 PM os]$ cd wk7

[s2022103042@centos8-linux Wed May 15 05:53 PM wk7]$ ls

**ex1** **ex2** **ex3** **ex4** **ex5** **ex6** **ex7** fifo p1.c p2.c p3.c p4.c p5.c p6.c p7.c **sp1** sp1.c **sp2** sp2.c

[s2022103042@centos8-linux Wed May 15 05:53 PM wk7]$ cat p1.c

#include <stdio.h>

#include <unistd.h>

int main() {

int pipe\_fd[2];

pid\_t pid;

char buffer[20];

pipe(pipe\_fd);

pid = fork();

if (pid == 0) {

close(pipe\_fd[1]);

read(pipe\_fd[0], buffer, sizeof(buffer));

printf("Child received: %s\n", buffer);

close(pipe\_fd[0]);

} else { // Parent process

close(pipe\_fd[0]);

write(pipe\_fd[1], "Hello, child!", 13);

printf("parent process written\n");

close(pipe\_fd[1]);

}

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:53 PM wk7]$ cat p2.c

#include<stdio.h>

#include<unistd.h>

int main() {

int pipefds[2];

int returnstatus;

char writemessages[2][20]={"Hi", "Hello"};

char readmessage[20];

returnstatus = pipe(pipefds);

if (returnstatus == -1) {

printf("Unable to create pipe\n");

return 1;

}

printf("Writing to pipe - Message 1 is %s\n", writemessages[0]);

write(pipefds[1], writemessages[0], sizeof(writemessages[0]));

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Reading from pipe – Message 1 is %s\n", readmessage);

printf("Writing to pipe - Message 2 is %s\n", writemessages[1]);

write(pipefds[1], writemessages[1], sizeof(writemessages[0]));

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Reading from pipe – Message 2 is %s\n", readmessage);

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:54 PM wk7]$ cat p3.c

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int main() {

mkfifo("fifo", 0666);

int fd1 = open("fifo", O\_RDWR);

int fd2 = open("fifo", O\_RDWR);

pid\_t pid = fork();

if (pid == 0) {

char msg[20];

read(fd1, msg, sizeof(msg));

printf("Child received: %s\n", msg);

close(fd1);

write(fd2, "Hello, parent!", 14);

close(fd2);

} else {

write(fd1, "Hello, child!", 13);

close(fd1);

wait(NULL);

char msg[20];

read(fd2, msg, sizeof(msg));

printf("Parent received: %s\n", msg);

close(fd2);

}

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:54 PM wk7]$ cat p4.c

#include<stdio.h>

#include<unistd.h>

int main() {

int pipefds1[2], pipefds2[2];

int returnstatus1, returnstatus2;

int pid;

char pipe1writemessage[20] = "Hi";

char pipe2writemessage[20] = "Hello";

char readmessage[20];

returnstatus1 = pipe(pipefds1);

if (returnstatus1 == -1) {

printf("Unable to create pipe 1 \n");

return 1;

}

returnstatus2 = pipe(pipefds2);

if (returnstatus2 == -1) {

printf("Unable to create pipe 2 \n");

return 1;

}

pid = fork();

if (pid != 0) {

close(pipefds1[0]); // Close the unwanted pipe1 read side

close(pipefds2[1]); // Close the unwanted pipe2 write side

printf("In Parent: Writing to pipe 1 – Message is %s\n", pipe1writemessage);

write(pipefds1[1], pipe1writemessage, sizeof(pipe1writemessage));

read(pipefds2[0], readmessage, sizeof(readmessage));

printf("In Parent: Reading from pipe 2 – Message is %s\n", readmessage);

} else { //child process

close(pipefds1[1]); // Close the unwanted pipe1 write side

close(pipefds2[0]); // Close the unwanted pipe2 read side

read(pipefds1[0], readmessage, sizeof(readmessage));

printf("In Child: Reading from pipe 1 – Message is %s\n", readmessage);

printf("In Child: Writing to pipe 2 – Message is %s\n", pipe2writemessage);

write(pipefds2[1], pipe2writemessage, sizeof(pipe2writemessage));

}

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:54 PM wk7]$ cat p5.c

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/wait.h>

#define NUM\_PIPES 3 // Define the number of pipes

int main() {

int fds[NUM\_PIPES][2]; // Array to hold file descriptors for multiple pipes

char buffer[100];

pid\_t p;

for (int i = 0; i < NUM\_PIPES; i++) {

if (pipe(fds[i]) == -1) {

perror("pipe");

return 1;

}

}

p = fork();

if (p > 0) { // Parent process

printf("Parent Passing value to child\n");

for (int i = 0; i < NUM\_PIPES; i++) {

write(fds[i][1], "hello\n", 6);

}

} else if (p == 0) { // Child process

printf("Child printing received value\n");

for (int i = 0; i < NUM\_PIPES; i++) {

read(fds[i][0], buffer, 100);

printf("Received value from pipe %d: %s", i, buffer);

}

} else { // Error handling

perror("fork");

return 1;

}

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:54 PM wk7]$ cat p6.c

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

int main() {

int pipe\_fd[2];

pipe(pipe\_fd);

pid\_t pid = fork();

if (pid == 0) { // Child process

close(pipe\_fd[0]); // Close read end

dup2(pipe\_fd[1], STDOUT\_FILENO); // Redirect stdout to pipe

close(pipe\_fd[1]); // Close write end

execl("/bin/ls", "ls", NULL); // Execute ls command

perror("execl");

exit(EXIT\_FAILURE);

} else { // Parent process

close(pipe\_fd[1]); // Close write end

dup2(pipe\_fd[0], STDIN\_FILENO); // Redirect stdin to pipe

close(pipe\_fd[0]); // Close read end

execl("/usr/bin/wc", "wc", "-l", NULL); // Execute wc -l command

perror("execl");

exit(EXIT\_FAILURE);

}

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:54 PM wk7]$ cat p7.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#define NUM\_COMMANDS 3

int main() {

int child\_pipes[NUM\_COMMANDS][2]; // Pipes for communication with childre// Create pipes

for (int i = 0; i < NUM\_COMMANDS; ++i) {

if (pipe(child\_pipes[i]) == -1) {

perror("pipe");

exit(EXIT\_FAILURE);

}

}// Fork child processes

for (int i = 0; i < NUM\_COMMANDS; ++i) {

pid\_t pid = fork();

if (pid == -1) {

perror("fork");

exit(EXIT\_FAILURE);

} else if (pid == 0) { // Child process // Close read end in child

close(child\_pipes[i][0]);// Redirect stdout to the write end of the pipe

dup2(child\_pipes[i][1], STDOUT\_FILENO);// Execute a command based on the child index

switch (i) {

case 0:

execlp("ls", "ls", "-l", NULL);

break;

case 1:

execlp("pwd", "pwd", NULL);

break;

case 2:

execlp("date", "date", NULL);

break;

default:

fprintf(stderr, "Unknown command index\n");

exit(EXIT\_FAILURE);

}

}

} // Parent process

for (int i = 0; i < NUM\_COMMANDS; ++i) {// Close write end in parent

close(child\_pipes[i][1]);// Read the output from the child

char buffer[1024];

ssize\_t bytes\_read;

printf("Output from child %d:\n", i + 1);

while ((bytes\_read = read(child\_pipes[i][0], buffer, sizeof(buffer))) != 0) {

if (bytes\_read == -1) {

perror("read");

exit(EXIT\_FAILURE);

}

write(STDOUT\_FILENO, buffer, bytes\_read);

}// Close read end in parent

close(child\_pipes[i][0]);

}

}

[s2022103042@centos8-linux Wed May 15 05:54 PM wk7]$ cat p8.c

cat: p8.c: No such file or directory

[s2022103042@centos8-linux Wed May 15 05:54 PM wk7]$ ls

**ex1** **ex2** **ex3** **ex4** **ex5** **ex6** **ex7** fifo p1.c p2.c p3.c p4.c p5.c p6.c p7.c **sp1** sp1.c **sp2** sp2.c

[s2022103042@centos8-linux Wed May 15 05:54 PM wk7]$ cat sp1.c

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

void fibo(int n)

{

int prev=0,curr=1,next;

printf("Fibonacci upto %d is:",n);

printf("%d, ",prev);

while(curr<=n){

printf("%d, ",curr);

next=prev+curr;

prev=curr;

curr=next;

}

printf("\n");

}

int main(){

int n,pipefd[2];

pid\_t pid;

if(pipe(pipefd)==-1){

perror("PIPE");

exit(EXIT\_FAILURE);

}

pid=fork();

if(pid==-1){

perror("FORK");

exit(EXIT\_FAILURE);

}

if(pid==0)

{

close(pipefd[1]);

read(pipefd[0],&n,sizeof(n));

close(pipefd[0]);

fibo(n);

}

else{

close(pipefd[0]);

printf("Enter the value of n:");

scanf("%d",&n);

write(pipefd[1], &n,sizeof(n));

close(pipefd[1]);

}

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:54 PM wk7]$ cat sp2.c

#include<stdio.h>

#include<stdlib.h>

struct TreeNode{

int val;

struct TreeNode \*left;

struct TreeNode \*right;

};

struct TreeNode\* newNode(int value){

struct TreeNode\* node=(struct TreeNode\*)malloc(sizeof(struct TreeNode));

node->val = value;

node->right = NULL;

node->left = NULL;

return node;

}

int count;

int countTriplets(struct TreeNode\* root){

if(root == NULL){

return 0;

}

if(root->right != NULL && root->left->left !=NULL && root->right->right != NULL){

int s1, s2, s3, s4;

s1 = root->val + root->left->val + root->left->left->val;

s2 = root->val + root->left->val + root->left->right->val;

s3 = root->val + root->right->val + root->right->left->val;

s4 = root->val + root->right->val + root->right->right->val;

if(s1 > 7){

count += 1;

}

if(s2 > 7){

count += 1;

}

if(s3 > 7){

count += 1;

}

if(s4 > 7){

count += 1;

}

}

countTriplets(root->left);

countTriplets(root->right);

return count;

}

int main(){

struct TreeNode \*root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

root->right->left = newNode(6);

root->right->right = newNode(7);

int tripletsCount = countTriplets(root);

printf("Number of triplets with sum greater than 7 : %d\n", tripletsCount);

return 0;

}

**Week 8**

[s2022103042@centos8-linux Wed May 15 05:54 PM wk7]$ cd ..

[s2022103042@centos8-linux Wed May 15 05:55 PM os]$ cd wk8

[s2022103042@centos8-linux Wed May 15 05:55 PM wk8]$ ls

a1.c a2.c a3.c a4.c a5.c a6.c **a.out** **ex1** **ex2** **ex3** **ex4** **ex5** **ex6** **s3** s3.c **s4** s4.c **sp1** sp1.c **sp2** **sp2a** sp2a.c sp2.c

[s2022103042@centos8-linux Wed May 15 05:55 PM wk8]$ cat a1.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#define SHM\_SIZE 1024

int main() {

key\_t key = ftok("shmfile", 65);

int shmid = shmget(key, SHM\_SIZE, 0666|IPC\_CREAT);

char \*str = (char\*) shmat(shmid, (void\*)0, 0);

printf("Enter data to write to shared memory: ");

fgets(str, SHM\_SIZE, stdin);

printf("Data written to shared memory: %s\n", str);

shmdt(str);

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:55 PM wk8]$ cat a2.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#define SHM\_SIZE 1024

int main() {

key\_t key = ftok("shmfile",65);

int shmid = shmget(key,SHM\_SIZE,0666|IPC\_CREAT);

char \*str = (char\*) shmat(shmid,(void\*)0,0);

printf("Data read from shared memory: %s\n",str);

shmdt(str);

shmctl(shmid,IPC\_RMID,NULL);

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:55 PM wk8]$ cat a3.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <unistd.h>

#include <sys/wait.h>

#define SHM\_SIZE sizeof(int) \* 2

int main() {

key\_t key = ftok("shmfile", 65);

int shmid = shmget(key, SHM\_SIZE, 0666|IPC\_CREAT);

int \*data = (int\*) shmat(shmid, (void\*)0, 0);

data[0] = 0; // Initialize counter

printf("Initial counter value: %d\n", data[0]);

int num\_processes = 3; // Number of child processes

for (int i = 0; i < num\_processes; i++) {

pid\_t pid = fork();

if (pid < 0) {

perror("Fork failed");

exit(EXIT\_FAILURE);

} else if (pid == 0) { // Child process

data[0]++; // Increment counter

data[data[0]] = getpid(); // Write PID to shared memory

printf("Child process %d wrote its PID to shared memory\n", getpid());

shmdt(data);

exit(EXIT\_SUCCESS);

}

}// Parent process waits for all child processes to finish

for (int i = 0; i < num\_processes; i++) {

wait(NULL);

}

printf("PIDs written to shared memory:\n");

printf("Final counter value: %d\n", data[0]);

for (int i = 1; i <= data[0]; i++) {

printf("%d\n", data[i]);

}

shmdt(data);

shmctl(shmid, IPC\_RMID, NULL); // Remove shared memory segment after use

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:55 PM wk8]$ cat a4.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <unistd.h>

#include <sys/wait.h>

#define SHM\_SIZE sizeof(int)

void perform\_calculation(int shmid) {

int \*result = (int\*) shmat(shmid, (void\*)0, 0);

int num1 ;

int num2 ;

printf("Enter two numbers(for addition) : ");

scanf("%d %d",&num1,&num2);

\*result = num1 + num2;

shmdt(result);

}

void read\_result\_from\_shared\_memory(int shmid) {

int \*result = (int\*) shmat(shmid, (void\*)0, 0);

printf("Result read from shared memory: %d\n", \*result);

shmdt(result);

}

int main() {

key\_t key = ftok("shmfile", 65);

int shmid = shmget(key, SHM\_SIZE, 0666|IPC\_CREAT);

int choice = 0;

while (choice != 3) {

printf("\n1. Perform calculation\n");

printf("2. Read result from shared memory\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

perform\_calculation(shmid);

printf("Calculation performed and result stored in shared memory.\n");

break;

case 2:

read\_result\_from\_shared\_memory(shmid);

break;

case 3:

printf("Exiting...\n");

break;

default:

printf("Invalid choice.\n");

break;

}

}

shmctl(shmid, IPC\_RMID, NULL); // Remove shared memory segment after use

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:55 PM wk8]$ cat a5.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#define SHM\_SIZE 1024

int main() {

key\_t key = ftok("shmfile", 65);

int shmid = shmget(key, SHM\_SIZE, 0666|IPC\_CREAT);

char \*str = (char\*) shmat(shmid, (void\*)0, 0);

printf("Enter a string to reverse: ");

fgets(str, SHM\_SIZE, stdin);

int len = strlen(str) - 1;

for (int i = 0; i < len / 2; i++) {

char temp = str[i];

str[i] = str[len - i - 1];

str[len - i - 1] = temp;

}

printf("Reversed string stored in shared memory: %s\n", str);

shmdt(str);

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:55 PM wk8]$ cat a6.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#define N 3

#define SHM\_SIZE sizeof(int) \* N \* N \* 3

void write\_matrices\_to\_shared\_memory(int shmid) {

int \*matrices = (int\*) shmat(shmid, (void\*)0, 0);

printf("Enter elements of matrix A (%dx%d):\n", N, N);

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

printf("A[%d][%d]: ", i, j);

scanf("%d", &matrices[i \* N + j]);

}

}

printf("Enter elements of matrix B (%dx%d):\n", N, N);

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

printf("B[%d][%d]: ", i, j);

scanf("%d", &matrices[N \* N + i \* N + j]);

}

}

shmdt(matrices);

}

void read\_result\_from\_shared\_memory(int shmid) {

int \*result = (int\*) shmat(shmid, (void\*)0, 0);

printf("Matrix multiplication result read from shared memory:\n");

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

printf("%d\t", result[2\*N\*N+i \* N + j]);

}

printf("\n");

}

shmdt(result);

}

void multiply\_matrices(int shmid) {

int \*matrices = (int\*) shmat(shmid, (void\*)0, 0);

int A[N][N], B[N][N], result[N][N];

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

A[i][j] = matrices[i \* N + j];

B[i][j] = matrices[N \* N + i \* N + j];

}

}

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

result[i][j] = 0;

for (int k = 0; k < N; k++) {

result[i][j] += A[i][k] \* B[k][j];

}

}

}

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

matrices[2 \* N \* N + i \* N + j] = result[i][j];

}

}

shmdt(matrices);

}

int main() {

key\_t key = ftok("shmfile", 65);

int shmid = shmget(key, SHM\_SIZE, 0666|IPC\_CREAT);

int choice = 0;

while (choice != 4) {

printf("\n1. Write matrices to shared memory\n");

printf("2. Read result from shared memory\n");

printf("3. Multiply matrics.\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

write\_matrices\_to\_shared\_memory(shmid);

printf("Matrices written to shared memory.\n");

break;

case 2:

read\_result\_from\_shared\_memory(shmid);

break;

case 3:

multiply\_matrices(shmid);

printf("Matrics multiplied.\n");

break;

case 4:

printf("Exiting...\n");

break;

default:

printf("Invalid choice.\n");

break;

}

}

shmctl(shmid, IPC\_RMID, NULL); // Remove shared memory segment after use

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:55 PM wk8]$ cat sp1.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <unistd.h>

#include <sys/wait.h>

#define SHM\_SIZE sizeof(int)

void perform\_calculation(int shmid) {

int \*result = (int\*) shmat(shmid, (void\*)0, 0);

int num1 ;

printf("Enter a numbers: ");

scanf("%d",&num1);

\*result = num1;

shmdt(result);

}

void read\_result\_from\_shared\_memory(int shmid) {

int \*result = (int\*) shmat(shmid, (void\*)0, 0);

for(int i=1;i<=10;i++){

printf("%dX%d=%d\n",i,\*result,i\*(\*result));

}

shmdt(result);

}

int main() {

key\_t key = ftok("shmfile", 65);

int shmid = shmget(key, SHM\_SIZE, 0666|IPC\_CREAT);

int choice = 0;

while (choice != 3) {

printf("\n1. Perform calculation\n");

printf("2. Read result from shared memory\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

perform\_calculation(shmid);

printf("value stored in shared memory.\n");

break;

case 2:

read\_result\_from\_shared\_memory(shmid);

break;

case 3:

printf("Exiting...\n");

break;

default:

printf("Invalid choice.\n");

break;

}

}

shmctl(shmid, IPC\_RMID, NULL); // Remove shared memory segment after use

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:55 PM wk8]$ cat sp2.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#define SHM\_SIZE 1024

int main() {

key\_t key1 = ftok("shmfile", 65);

int shmid1 = shmget(key1, SHM\_SIZE, 0666|IPC\_CREAT);

char \*str1 = (char\*) shmat(shmid1, (void\*)0, 0);

printf("Enter data to write to shared memory: ");

fgets(str1, SHM\_SIZE, stdin);

printf("Data written to shared memory: %s\n", str1);

shmdt(str1);

key\_t key2 = ftok("shmfile", 95);

int shmid2 = shmget(key2, SHM\_SIZE, 0666|IPC\_CREAT);

char \*str2 = (char\*) shmat(shmid2, (void\*)0, 0);

printf("Enter data to write to shared memory: ");

fgets(str2, SHM\_SIZE, stdin);

printf("Data written to shared memory: %s\n", str2);

shmdt(str2);

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:55 PM wk8]$ cat sp2a.c

#include <stdio.h>

#include<string.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#define SHM\_SIZE 1024

int main() {

key\_t key1 = ftok("shmfile",65);

int shmid1 = shmget(key1,SHM\_SIZE,0666|IPC\_CREAT);

char \*str1 = (char\*) shmat(shmid1,(void\*)0,0);

printf("Data read from shared memory: %s\n",str1);

printf("length of string 1 is %d\n",strlen(str1));

shmdt(str1);

shmctl(shmid1,IPC\_RMID,NULL);

key\_t key2 = ftok("shmfile",95);

int shmid2 = shmget(key2,SHM\_SIZE,0666|IPC\_CREAT);

char \*str2 = (char\*) shmat(shmid2,(void\*)0,0);

printf("Data read from shared memory: %s\n",str2);

printf("length of string 2 is %d\n",strlen(str2));

shmdt(str2);

shmctl(shmid2,IPC\_RMID,NULL);

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:55 PM wk8]$ cat s3.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <unistd.h>

#include <sys/wait.h>

#define SHM\_SIZE sizeof(int)

void perform\_calculation(int shmid) {

int \*result = (int\*) shmat(shmid, (void\*)0, 0);

int num1 ;

printf("Enter a mark : ");

scanf("%d",&num1);

\*result = num1;

shmdt(result);

}

void read\_result\_from\_shared\_memory(int shmid) {

int \*result = (int\*) shmat(shmid, (void\*)0, 0);

printf("Result read from shared memory:");

if((\*result)>90)

printf("O grade");

else if((\*result)>75)

printf("A grade");

else if ((\*result)>50)

printf("B Grade");

else

printf("Fail");

shmdt(result);

}

int main() {

key\_t key = ftok("shmfile", 65);

int shmid = shmget(key, SHM\_SIZE, 0666|IPC\_CREAT);

int choice = 0;

while (choice != 3) {

printf("\n1.MARK ENTRY\n");

printf("2. Read grade result from shared memory\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

perform\_calculation(shmid);

printf("Mark stored in shared memory.\n");

break;

case 2:

read\_result\_from\_shared\_memory(shmid);

break;

case 3:

printf("Exiting...\n");

break;

default:

printf("Invalid choice.\n");

break;

} }

}

[s2022103042@centos8-linux Wed May 15 05:56 PM wk8]$ cat s4.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <unistd.h>

#include <sys/wait.h>

#define SHM\_SIZE sizeof(int)

void perform\_calculation(int shmid) {

int \*result = (int\*) shmat(shmid, (void\*)0, 0);

int num1,num2,num3 ;

printf("Enter a numbers(for pattern) : ");

scanf("%d %d %d",&num1,&num2,&num3);

\*result = num3\*100+num2\*10+num1;

shmdt(result);

}

void read\_result\_from\_shared\_memory(int shmid) {

int \*result = (int\*) shmat(shmid, (void\*)0, 0);

printf("Result read from shared memory:");

printf("\n");

for(int i=1;i<=((\*result)%10);i++)

printf("\tx");

printf("\n");

for (int i=1;i<=((\*result)/10)%10;i++)

printf("\ty");

printf("\n");

for (int i=1;i<=(\*result)/100;i++)

printf("\tz");

shmdt(result);

}

int main() {

key\_t key = ftok("shmfile", 65);

int shmid = shmget(key, SHM\_SIZE, 0666|IPC\_CREAT);

int choice = 0;

while (choice != 3) {

printf("\n1. pattern heading\n");

printf("2. Read result from shared memory\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

perform\_calculation(shmid);

printf("Result stored in shared memory.\n");

break;

case 2:

read\_result\_from\_shared\_memory(shmid);

break;

case 3:

printf("Exiting...\n");

break;

default:

printf("Invalid choice.\n");

break;

} }

}

**Week 9**

[s2022103042@centos8-linux Wed May 15 05:56 PM wk8]$ cd ..

[s2022103042@centos8-linux Wed May 15 05:56 PM os]$ cd wk9

[s2022103042@centos8-linux Wed May 15 05:56 PM wk9]$ ls

**a.out** **fcfs** fcfs.c **prio** prio.c **rr** rr1.c rr.c **sjf** sjf1.c sjf.c **sp1** sp1.c **sp2** sp2.c **srtf** srtf.c

[s2022103042@centos8-linux Wed May 15 05:56 PM wk9]$ cat fcfs.c

#include <stdio.h>

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void bubbleSort(int pid[], int bt[], int n) {

int i, j;

for (i = 0; i < n-1; i++) {

for (j = 0; j < n-i-1; j++) {

if (pid[j] > pid[j+1]) {

swap(&pid[j], &pid[j+1]);

swap(&bt[j], &bt[j+1]);

}

}

}

}

int main() {

int pid[15];

int bt[15];

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

printf("Enter arrival time of all the processes: ");

for(int i = 0; i < n; i++) {

scanf("%d", &pid[i]);

}

printf("Enter burst time of all the processes: ");

for(int i = 0; i < n; i++) {

scanf("%d", &bt[i]);

}

bubbleSort(pid, bt, n);

int i, wt[n];

wt[0] = 0;

for(i = 1; i < n; i++) {

wt[i] = bt[i - 1] + wt[i - 1];

}

printf("Arrival Time Burst Time Waiting Time Turnaround Time\n");

float twt = 0.0;

float tat = 0.0;

for(i = 0; i < n; i++) {

printf("%d\t\t", pid[i]);

printf("%d\t\t", bt[i]);

printf("%d\t\t", wt[i]);

printf("%d\t\t", bt[i] + wt[i]);

printf("\n");

twt += wt[i];

tat += (wt[i] + bt[i]);

}

float att, awt;

awt = twt / n;

att = tat / n;

printf("Avg. waiting time= %f\n", awt);

printf("Avg. turnaround time= %f", att);

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:56 PM wk9]$ cat sjf.c

#include<stdio.h>

int main()

{

int i,n,p[10]={1,2,3,4,5,6,7,8,9,10},min,k=1,btime=0;

int bt[10],temp,j,at[10],wt[10],tt[10],ta=0,sum=0;

float wavg=0,tavg=0,tsum=0,wsum=0;

printf(" -------Shortest Job First Scheduling ( NP )-------\n");

printf("\nEnter the No. of processes :");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("\tEnter the burst time of %d process :",i+1);

scanf(" %d",&bt[i]);

printf("\tEnter the arrival time of %d process :",i+1);

scanf(" %d",&at[i]);

}

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

if(at[i]<at[j])

{

temp=p[j];

p[j]=p[i];

p[i]=temp;

temp=at[j];

at[j]=at[i];

at[i]=temp;

temp=bt[j];

bt[j]=bt[i];

bt[i]=temp;

}

}

}

for(j=0;j<n;j++)

{

btime=btime+bt[j];

min=bt[k];

for(i=k;i<n;i++)

{

if (btime>=at[i] && bt[i]<min)

{

temp=p[k];

p[k]=p[i];

p[i]=temp;

temp=at[k];

at[k]=at[i];

at[i]=temp;

temp=bt[k];

bt[k]=bt[i];

bt[i]=temp;

}

}

k++;

}

wt[0]=0;

sum=at[0];

for(i=1;i<n;i++)

{

sum=sum+bt[i-1];

wt[i]=sum-at[i];

wsum=wsum+wt[i];

}

wavg=(wsum/n);

for(i=0;i<n;i++)

{

ta=ta+bt[i];

tt[i]=ta-at[i];

tsum=tsum+tt[i];

}

tavg=(tsum/n);

printf("");

printf("\n RESULT:-");

printf("\nProcess\t Burst\t Arrival\t Waiting\t Turn-around" );

for(i=0;i<n;i++)

{

printf("\n p%d\t %d\t %d\t\t %d\t\t\t%d",p[i],bt[i],at[i],wt[i],tt[i]);

}

printf("\n\nAVERAGE WAITING TIME : %f",wavg);

printf("\nAVERAGE TURN AROUND TIME : %f",tavg);

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:56 PM wk9]$ cat srtf.c

#include <stdio.h>

struct process {

int pid;

int at;

int bt;

int st;

int ct;

int tat;

int wt;

int rt;

};

int main() {

int x;

struct process p[100];

float avg\_tat;

float avg\_wt;

int total\_tat = 0;

int total\_wt = 0;

int burst\_remaining[100] = {0};

int is\_completed[100] = {0};

printf("Enter the number of processes: ");

scanf("%d", &x);

for (int i = 0; i < x; i++) {

printf("Enter arrival time of process %d: ", i + 1);

scanf("%d", &p[i].at);

printf("Enter burst time of process %d: ", i + 1);

scanf("%d", &p[i].bt);

p[i].pid = i + 1;

burst\_remaining[i] = p[i].bt;

}

int current\_time = 0;

int completed = 0;

int prev = 0;

while (completed != x) {

int idx = -1;

int mn = 10000000;

for (int i = 0; i < x; i++) {

if (p[i].at <= current\_time && is\_completed[i] == 0) {

if (burst\_remaining[i] < mn) {

mn = burst\_remaining[i];

idx = i;

}

if (burst\_remaining[i] == mn) {

if (p[i].at < p[idx].at) {

mn = burst\_remaining[i];

idx = i;

}

}

}

}

if (idx != -1) {

if (burst\_remaining[idx] == p[idx].bt) {

p[idx].st = current\_time;

}

burst\_remaining[idx]--;

current\_time++;

if (burst\_remaining[idx] == 0) {

p[idx].ct = current\_time;

p[idx].tat = p[idx].ct - p[idx].at;

p[idx].wt = p[idx].tat - p[idx].bt;

total\_tat += p[idx].tat;

total\_wt += p[idx].wt;

is\_completed[idx] = 1;

completed++;

}

} else {

current\_time++;

}

}

avg\_tat = (float) total\_tat / x;

avg\_wt = (float) total\_wt / x;

printf("\nProcess\tTurnaround Time\tWaiting Time\n");

for (int i = 0; i < x; i++) {

printf("%d\t%d\t\t%d\n", p[i].pid, p[i].tat, p[i].wt);

}

printf("Average Turnaround Time: %.2f\n", avg\_tat);

printf("Average Waiting Time: %.2f\n", avg\_wt);

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:56 PM wk9]$ cat prio.c

#include <stdio.h>

#include <stdlib.h>// Define a structure to represent a process

struct Process {

int id; // Process ID

int bt; // Burst time

int pr; // Priority

int wt; // Waiting time

int tat; // Turnaround time

};// Comparator function for sorting processes based on arrival time and priority

int compare(const void \*a, const void \*b) {

const struct Process \*p1 = (const struct Process \*)a;

const struct Process \*p2 = (const struct Process \*)b;

return p1->pr - p2->pr; // If arrival times are equal, sort by priority (descending)

}

int main() {

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

struct Process p[n];// Input process details

for (int i = 0; i < n; i++) {

printf("Enter Process id, Burst time & Priority for Process %d: ", i + 1);

scanf("%d %d %d", &p[i].id, &p[i].bt, &p[i].pr);

}// Sort processes based on arrival time and priority

qsort(p, n, sizeof(struct Process), compare);// Calculate waiting time and turnaround time

p[0].wt = 0;

p[0].tat = p[0].bt;

int time = p[0].bt;

double tatSum = p[0].tat;

double wtSum = p[0].wt;

for (int i = 1; i < n; ++i) {

time += p[i].bt;

p[i].tat = time;

p[i].wt = p[i].tat - p[i].bt;

tatSum += p[i].tat;

wtSum += p[i].wt;

}// Sort processes by ID before printing

qsort(p, n, sizeof(struct Process), compare);

printf("\nID\tTAT\tWT\n\n");

for (int i = 0; i < n; ++i) {

printf("%d\t%d\t%d\n", p[i].id, p[i].tat, p[i].wt);

}

printf("\nAverage Waiting Time -- %.2lf", wtSum / n);

printf("\nAverage Turnaround Time -- %.2lf", tatSum / n);

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:57 PM wk9]$ cat rr.c

#include <stdio.h>

#define MAX\_PROCESSES 10

#define TIME\_QUANTUM 2

struct Process {

int pid;

int burst;

int remaining;

int arrival;

int waiting;

int turnaround;

};

void calculateTimes(struct Process processes[], int n, int quantum) {

int currentTime = 0;

int remainingProcesses = n;

while (remainingProcesses > 0) {

for (int i = 0; i < n; i++) {

if (processes[i].remaining > 0) {

if (processes[i].remaining > quantum) {

currentTime += quantum;

processes[i].remaining -= quantum;

} else {

currentTime += processes[i].remaining;

processes[i].waiting = currentTime - processes[i].arrival - processes[i].burst;

processes[i].turnaround = currentTime - processes[i].arrival;

processes[i].remaining = 0;

remainingProcesses--;

}

}

}

}

}int main() {

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

struct Process processes[MAX\_PROCESSES];

for (int i = 0; i < n; i++) {

printf("Enter arrival time and burst time for process %d: ", i + 1);

scanf("%d %d", &processes[i].arrival, &processes[i].burst);

processes[i].pid = i + 1;

processes[i].remaining = processes[i].burst;

}

calculateTimes(processes, n, TIME\_QUANTUM);

printf("PID\tWaiting Time\tTurnaround Time\n");

for (int i = 0; i < n; i++) {

printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].waiting, processes[i].turnaround);

}

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:57 PM wk9]$ cat sp1.c

#include <stdio.h>

#define context 1

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void bubbleSort(int pid[], int bt[], int n) {

int i, j;

for (i = 0; i < n-1; i++) {

for (j = 0; j < n-i-1; j++) {

if (pid[j] > pid[j+1]) {

swap(&pid[j], &pid[j+1]);

swap(&bt[j], &bt[j+1]);

}

}

}

}

int main() {

int pid[15];

int bt[15];

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

printf("Enter arrival time of all the processes: ");

for(int i = 0; i < n; i++) {

scanf("%d", &pid[i]);

}

printf("Enter burst time of all the processes: ");

for(int i = 0; i < n; i++) {

scanf("%d", &bt[i]);

}

bubbleSort(pid, bt, n);

int i, wt[n];

wt[0] = 0;

for(i = 1; i < n; i++) {

wt[i] = bt[i - 1] + wt[i - 1]+context;

}

printf("Arrival Time Burst Time Waiting Time Turnaround Time\n");

float twt = 0.0;

float tat = 0.0;

for(i = 0; i < n; i++) {

printf("%d\t\t", pid[i]);

printf("%d\t\t", bt[i]);

printf("%d\t\t", wt[i]);

printf("%d\t\t", bt[i] + wt[i]);

printf("\n");

twt += wt[i];

tat += (wt[i] + bt[i]);

}

float att, awt;

awt = twt / n;

att = tat / n;

printf("Avg. waiting time= %f\n", awt);

printf("Avg. turnaround time= %f", att);

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:57 PM wk9]$ cat sp2.c

#include <stdio.h>

struct process {

int id;

int burst\_time;

int priority;

int remaining;

int waiting;

int turnaround;

};

void calculateTimes1(struct process queue1[], int n, int quantum) {

int currentTime = 0;

int remainingProcesses = n;

while (remainingProcesses > 0) {

for (int i = 0; i < n; i++) {

if (queue1[i].remaining > 0) {

if (queue1[i].remaining > quantum) {

currentTime += quantum;

queue1[i].remaining -= quantum;

} else {

currentTime += queue1[i].remaining;

queue1[i].waiting = currentTime - queue1[i].burst\_time;

queue1[i].turnaround = currentTime;

queue1[i].remaining = 0;

remainingProcesses--;

}

}

}

}

}

void calculateTimes2(struct process queue2[], int n, int quantum) {

int currentTime = 0;

int remainingProcesses = n;

while (remainingProcesses > 0) {

for (int i = 0; i < n; i++) {

if (queue2[i].remaining > 0) {

if (queue2[i].remaining > quantum) {

currentTime += quantum;

queue2[i].remaining -= quantum;

} else {

currentTime += queue2[i].remaining;

queue2[i].waiting = currentTime - queue2[i].burst\_time;

queue2[i].turnaround = currentTime;

queue2[i].remaining = 0;

remainingProcesses--;

}

}

}

}

}

void multi\_level\_queue(struct process p[], int n) {

struct process queue1[n], queue2[n], queue3[n];

int front1 = -1, rear1 = -1, front2 = -1, rear2 = -1, front3 = -1, rear3 = -1;

for (int i = 0; i < n; i++) {

if (p[i].priority == 1) {

if (front1 == -1)

front1 = 0;

rear1++;

queue1[rear1] = p[i];

}

else if (p[i].priority == 2) {

if (front2 == -1)

front2 = 0;

rear2++;

queue2[rear2] = p[i];

}

else {

if (front3 == -1)

front3 = 0;

rear3++;

queue3[rear3] = p[i];

}

}

printf("Queue 1 (Priority 1):\n");

int total\_waiting\_time = 0, total\_turnaround\_time = 0,n1=0;

for (int i = front1; i <= rear1; i++) {

n1++;

}

for (int i = 0; i < n1; i++) {

queue1[i].remaining = queue1[i].burst\_time;

}

calculateTimes1(queue1, n1, 3);

for (int i = front1; i <= rear1; i++) {

printf("Process ID: %d, Burst Time: %d\n", queue1[i].id, queue1[i].burst\_time);

total\_turnaround\_time += queue1[i].turnaround;

printf("Waiting Time: %d, Turnaround Time: %d\n", total\_waiting\_time, total\_turnaround\_time);

total\_waiting\_time += queue1[i].burst\_time;

}

printf("Queue 2 (Priority 2):\n");

int n2=0;

for (int i = front2; i <= rear2; i++) {

n2++;

}

for (int i = 0; i < n2; i++) {

queue2[i].remaining = queue2[i].burst\_time;

}

calculateTimes2(queue2, n2, 9);

for (int i = front2; i <= rear2; i++) {

printf("Process ID: %d, Burst Time: %d\n", queue1[i].id, queue1[i].burst\_time);

total\_turnaround\_time += queue2[i].turnaround;

printf("Waiting Time: %d, Turnaround Time: %d\n", total\_waiting\_time, total\_turnaround\_time);

total\_waiting\_time += queue2[i].burst\_time;

}

printf("Queue 3 (Priority 3):\n");

for (int i = front3; i <= rear3; i++) {

printf("Process ID: %d, Burst Time: %d\n", queue3[i].id, queue3[i].burst\_time);

total\_turnaround\_time += queue3[i].burst\_time;

printf("Waiting Time: %d, Turnaround Time: %d\n", total\_waiting\_time, total\_turnaround\_time);

total\_waiting\_time += queue3[i].burst\_time;

}

printf("Average Waiting Time: %.2f\n", (float)total\_waiting\_time / n);

printf("Average Turnaround Time: %.2f\n", (float)total\_turnaround\_time / n);

}

int main() {

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

struct process p[n];

for (int i = 0; i < n; i++) {

printf("Enter details for process %d (ID BurstTime Priority): ", i + 1);

scanf("%d %d %d", &p[i].id, &p[i].burst\_time, &p[i].priority);

}

multi\_level\_queue(p, n);

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:57 PM wk9]$ cat rr1.c

#include<stdio.h>

struct Process {

int pid;

int burst\_time;

int remaining\_time;

int arrival\_time;

int waiting\_time;

int turnaround\_time;

};

void calculateTimes(struct Process proc[], int n, int quantum) {

int remaining\_burst[n];

for (int i = 0; i < n; i++) {

remaining\_burst[i] = proc[i].burst\_time;

}

int t = 0;

while (1) {

int done = 1;

for (int i = 0; i < n; i++) {

if (remaining\_burst[i] > 0) {

done = 0;

if (remaining\_burst[i] > quantum) {

t += quantum;

remaining\_burst[i] -= quantum;

} else {

t += remaining\_burst[i];

proc[i].waiting\_time = t - proc[i].burst\_time - proc[i].arrival\_time;

remaining\_burst[i] = 0;

}

}

}

if (done == 1)

break;

}

for (int i = 0; i < n; i++) {

proc[i].turnaround\_time = proc[i].burst\_time + proc[i].waiting\_time;

}

}

void displayProcesses(struct Process proc[], int n) {

printf("Process ID Burst Time Arrival Time Waiting Time Turnaround Time\n");

for (int i = 0; i < n; i++) {

printf("%7d %12d %14d %14d %16d\n", proc[i].pid, proc[i].burst\_time, proc[i].arrival\_time,

proc[i].waiting\_time, proc[i].turnaround\_time);

}

}

int main() {

int n, quantum;

printf("Enter the number of processes: ");

scanf("%d", &n);

struct Process proc[n];

for (int i = 0; i < n; i++) {

printf("Enter the arrival time and burst time for process %d: ", i + 1);

scanf("%d %d", &proc[i].arrival\_time, &proc[i].burst\_time);

proc[i].pid = i + 1;

}

printf("Enter the time quantum: ");

scanf("%d", &quantum); calculateTimes(proc,n,quantum);

displayProcesses(proc,n);

return 0;

}

[s2022103042@centos8-linux Wed May 15 05:57 PM wk9]$ cat sjf1.c

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

struct Process {

int id;

int arrival\_time;

int burst\_time;

int waiting\_time;

int turnaround\_time;

};

int compareArrivalTime(const void \*a, const void \*b) {

return ((struct Process \*)a)->arrival\_time - ((struct Process \*)b)->arrival\_time;

}

int compareBurstTime(const void \*a, const void \*b) {

return ((struct Process \*)a)->burst\_time - ((struct Process \*)b)->burst\_time;

}

void findWaitingTime(struct Process processes[], int n) {

int currentTime = 0;

int remainingTime[n];

for (int i = 0; i < n; i++) {

remainingTime[i] = processes[i].burst\_time;

}

int completed = 0;

while (completed != n) {

int minBurstIndex = -1, minBurst = INT\_MAX;

for (int i = 0; i < n; i++) {

if (processes[i].arrival\_time <= currentTime && remainingTime[i] < minBurst && remainingTime[i] > 0) {

minBurst = remainingTime[i];

minBurstIndex = i;

}

}

if (minBurstIndex == -1) {

currentTime++;

continue;

}

remainingTime[minBurstIndex]--;

if (remainingTime[minBurstIndex] == 0) {

completed++;

processes[minBurstIndex].waiting\_time = currentTime + 1 - processes[minBurstIndex].arrival\_time - processes[minBurstIndex].burst\_ti

me;

if (processes[minBurstIndex].waiting\_time < 0) {

processes[minBurstIndex].waiting\_time = 0;

}

}

currentTime++;

}

}

void findTurnaroundTime(struct Process processes[], int n) {

for (int i = 0; i < n; i++) {

processes[i].turnaround\_time = processes[i].burst\_time + processes[i].waiting\_time;

}

}

void findAverageTime(struct Process processes[], int n) {

int total\_waiting\_time = 0, total\_turnaround\_time = 0;

findWaitingTime(processes, n);

findTurnaroundTime(processes, n);

printf("Process\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");

for (int i = 0; i < n; i++) {

total\_waiting\_time += processes[i].waiting\_time;

total\_turnaround\_time += processes[i].turnaround\_time;

printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].arrival\_time,

processes[i].burst\_time, processes[i].waiting\_time,

processes[i].turnaround\_time);

}

printf("\nAverage Waiting Time: %.2f\n", (float)total\_waiting\_time / n);

printf("Average Turnaround Time: %.2f\n", (float)total\_turnaround\_time / n);

}

int main() {

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

struct Process processes[n];

printf("Enter arrival time and burst time for each process:\n");

for (int i = 0; i < n; i++) {

processes[i].id = i + 1;

printf("Process %d: ", i + 1);

scanf("%d %d", &processes[i].arrival\_time, &processes[i].burst\_time);

}

qsort(processes, n, sizeof(struct Process), compareArrivalTime);

findAverageTime(processes, n);

return 0;

}

**Week 10**

[s2022103042@centos8-linux Wed May 15 06:17 PM os]$ cd wk10

[s2022103042@centos8-linux Wed May 15 06:17 PM wk10]$ ls

**a.out** **bb** bb.c **dp** dp.c **rw** rw.c **sp** sp.c

[s2022103042@centos8-linux Wed May 15 06:17 PM wk10]$ cat bb.c

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#define BUFFER\_SIZE 5 s

#define NUM\_ITEMS 10

sem\_t empty, full;

pthread\_mutex\_t mutex;

int buffer[BUFFER\_SIZE];

int in = 0, out = 0;

void \*producer(void \*arg) {

for (int i = 0; i < NUM\_ITEMS; i++) {

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

buffer[in] = i;

in = (in + 1) % BUFFER\_SIZE;

printf("Produced item %d\n", i);

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

}

pthread\_exit(NULL);

}

void \*consumer(void \*arg) {

for (int i = 0; i < NUM\_ITEMS; i++) {

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

int item = buffer[out];

out = (out + 1) % BUFFER\_SIZE;

printf("Consumed item %d\n", item);

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

}

pthread\_exit(NULL);

}

int main() {

pthread\_t producer\_thread, consumer\_thread;

pthread\_mutex\_init(&mutex, NULL);

sem\_init(&empty, 0, BUFFER\_SIZE);

sem\_init(&full, 0, 0);

pthread\_create(&producer\_thread, NULL, producer, NULL);

pthread\_create(&consumer\_thread, NULL, consumer, NULL);

pthread\_join(producer\_thread, NULL);

pthread\_join(consumer\_thread, NULL);

pthread\_mutex\_destroy(&mutex);

sem\_destroy(&empty);

sem\_destroy(&full);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:17 PM wk10]$ cat rw.c

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define NUM\_READERS 3

#define NUM\_WRITERS 2

sem\_t mutex, wrt;

int read\_count = 0;

int terminate = 0; // Variable to indicate when threads should terminate

void \*reader(void \*arg) {

while (!terminate) {

sem\_wait(&mutex);

read\_count++;

if (read\_count == 1) {

sem\_wait(&wrt);

}

sem\_post(&mutex);

printf("Reading data by reader %d\n", \*(int\*)arg);

sem\_wait(&mutex);

read\_count--;

if (read\_count == 0) {

sem\_post(&wrt);

}

sem\_post(&mutex);

usleep(1000000);

}

pthread\_exit(NULL); // Exit the thread when termination condition is met

}

void \*writer(void \*arg) {

while (!terminate) {

sem\_wait(&wrt);

printf("Writing data by writer %d\n", \*(int\*)arg);

sem\_post(&wrt);

usleep(2000000);

}

pthread\_exit(NULL); // Exit the thread when termination condition is met

}

int main() {

pthread\_t readers[NUM\_READERS], writers[NUM\_WRITERS];

sem\_init(&mutex, 0, 1);

sem\_init(&wrt, 0, 1);

int reader\_ids[NUM\_READERS];

int writer\_ids[NUM\_WRITERS];

for (int i = 0; i < NUM\_READERS; i++) {

reader\_ids[i] = i + 1;

pthread\_create(&readers[i], NULL, reader, &reader\_ids[i]);

}

for (int i = 0; i < NUM\_WRITERS; i++) {

writer\_ids[i] = i + 1;

pthread\_create(&writers[i], NULL, writer, &writer\_ids[i]);

}

sleep(10);

terminate = 1;

for (int i = 0; i < NUM\_READERS; i++) {

pthread\_join(readers[i], NULL);

}

for (int i = 0; i < NUM\_WRITERS; i++) {

pthread\_join(writers[i], NULL);

}

sem\_destroy(&mutex);

sem\_destroy(&wrt);

return 0;

}

[s2022103042@centos8-linux Wed May 15 06:17 PM wk10]$ cat dp.c

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define NUM\_PHILOSOPHERS 5

#define THINKING 0

#define HUNGRY 1

#define EATING 2

#define NUM\_CYCLES 5 // Number of cycles before termination

sem\_t mutex;

sem\_t forks[NUM\_PHILOSOPHERS];

int state[NUM\_PHILOSOPHERS];

int cycles\_completed = 0; // Counter for completed cycles

void test(int philosopher\_id) {

if (state[philosopher\_id] == HUNGRY &&

state[(philosopher\_id + 4) % NUM\_PHILOSOPHERS] != EATING &&

state[(philosopher\_id + 1) % NUM\_PHILOSOPHERS] != EATING) {

state[philosopher\_id] = EATING;

printf("Philosopher %d is eating\n", philosopher\_id);

sem\_post(&forks[philosopher\_id]);

}

}

void take\_forks(int philosopher\_id) {

sem\_wait(&mutex);

state[philosopher\_id] = HUNGRY;

printf("Philosopher %d is hungry\n", philosopher\_id);

test(philosopher\_id);

sem\_post(&mutex);

sem\_wait(&forks[philosopher\_id]);

}

void put\_forks(int philosopher\_id) {

sem\_wait(&mutex);

state[philosopher\_id] = THINKING;

printf("Philosopher %d finished eating and is thinking\n", philosopher\_id);

test((philosopher\_id + 4) % NUM\_PHILOSOPHERS);

test((philosopher\_id + 1) % NUM\_PHILOSOPHERS);

sem\_post(&mutex);

}

void \*philosopher(void \*arg) {

int philosopher\_id = \*(int \*)arg;

while (1) {

if (cycles\_completed == NUM\_CYCLES) {

break;

}

printf("Philosopher %d is thinking\n", philosopher\_id);

usleep(2000000); // Simulating thinking time

take\_forks(philosopher\_id);

usleep(1000000); // Simulating eating time

put\_forks(philosopher\_id);

if (philosopher\_id == NUM\_PHILOSOPHERS - 1) {

sem\_wait(&mutex);

cycles\_completed++;

sem\_post(&mutex);

}

}

pthread\_exit(NULL);

}

int main() {

pthread\_t philosophers[NUM\_PHILOSOPHERS];

sem\_init(&mutex, 0, 1);

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) {

sem\_init(&forks[i], 0, 1);

}

int philosopher\_ids[NUM\_PHILOSOPHERS];

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) {

philosopher\_ids[i] = i;

pthread\_create(&philosophers[i], NULL, philosopher, &philosopher\_ids[i]);

}

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) { pthread\_join(philosophers[i], NULL);

}

sem\_destroy(&mutex);

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) {

sem\_destroy(&forks[i]);

}

return 0;

}