TCS-502

Operating Systems Lab

TERM WORK

Name: Harsh Chaturvedi

Section: ML

Roll No: 2014669

Class Roll No: 29

Program 1

Objective

Demonstration of fork() System Call,

Solution

Single Fork

Code:

```c
#include <stdio.h>
#include <unistd.h>
int main()
{
    fork();
    printf("Hello, World!\n");
    return 0;
}
```

Result:

```
Hello, World!
Hello, World!
```

Multi Time Fork

Code:

```c
#include <stdio.h>
#include <unistd.h>
int main()
{
    fork();
    printf("Un\n");
    fork();
    printf("Dos\n");
    fork();
    printf("Tres\n");
    return 0;
}
```

Result:

```
Un
Dos
Un
Tres
Dos
```

Dos
Tres
Dos
Tres
Tres
Tres
Tres
Tres
Tres

Program 2

Objective

Parent Process Computes the Sum of Even and Child Process Computes the sum of Odd Numbers using fork .

Solution

Code:

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

#define max 20

int main()
{
    pid_t pid;
    int a[max], n, sum = 0, i, status;
    printf("Enter no of terms in array: >");
    scanf("%d", &n);
    printf("Enter values in array: >");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
        pid = fork();
        wait(&status);
        if (pid == 0)
        {
            for (i = 0; i < n; i++)
            {
                if (a[i] % 2 == 0)
                {
                    sum = sum + a[i];
                }
                printf("Sum of even nos = %d\n", sum);
            }
            exit(0);
        }
        else
        {
```

```
                for (i = 0; i < n; i++)
                {
                    if (a[i] % 2 != 0)
                    {
                        sum = sum + a[i];
                    }
                }
            }
        }
        return 0;
    }
```

Result:

```
Enter no of terms in array: > 6
Enter values in array: > 1 2 3 4 5 6
Sum of even nos = 12
```

Program 3

Objective

Demonstration of wait() System Call.

Solution

Code:

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

int main()
{
    pid_t pid;
    int status;
    pid = fork();
    if (pid == 0)
    {
        printf("Child\n");
        exit(0);
    }
    else
    {
        wait(&status);
        printf("Parent\n");
        printf("Child PID: %d\n", pid);
    }
    return 0;
}
```

Result:

```
Child
Parent
Child PID: 13357
```

Program 4

Objective

Implementation of Orphan Process & Zombie Process

Solution

Orphan Process

Code:

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

int main()
{
    pid_t pid;
    pid = fork();
    if (pid == 0)
    {
        sleep(6);
        printf("\n :: Child. (PID = %d And PPID = %d)", getpid(),
getppid());
    }
    else
    {
        printf(":: Parent. (Child PID = %d And my PID =%d)", pid,
getpid());
    }
    printf("\nTerminating PID = %d\n", getpid());
    return 0;
}
```

Result:

```
:: Parent. (Child PID = 2261 And my PID =2260)
Terminating PID = 2260
```

Zombie Process

Code:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
    pid_t pid;
    pid = fork();
    if (pid != 0)
    {
        while (1)
```

```
            sleep(50);
        }
        else
        {
            exit(0);
        }
    }
```

Result:

```
flamingarch@Harshs-MacBook-Pro:~$ ./a.out & [1]
2761
flamingarch@Harshs-MacBook-Pro:~$ ps
PID TTY    TIME CMD
2611 pts/0 00:00:00 bash
2761 pts/0 00:00:00 pb1
2762 pts/0 00:00:00 pb1 <defunct>
2763 pts/0 00:00:00
flamingarch@Harshs-MacBook-Pro:~$ kill 2761
flamingarch@Harshs-MacBook-Pro:~$ psPID TTYTIME CMD
2611 pts/0 00:00:00 bash
2764 pts/0 00:0y0:00 ps
[1]+ Terminated /pb1
flamingarch@Harshs-MacBook-Pro:~$
```

Program 5

Objective

Implementation of PIPE

Solution

Code:

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

int main()
{
    pid_t pid;
    char arr[100], str[100];
    int fd[2], nbr, nbw;
    pipe(fd);
    pid = fork();
    if (pid == 0)
    {
        printf("\nEnter a string: ");
        gets(str);
        nbw = write(fd[1], str, strlen(str));
        printf("Child wrote %d bytes\n", nbw);
        exit(0);
    }
    else
    {
        nbr = read(fd[0], arr, sizeof(arr));
        arr[nbr] = '\0';
        printf("Parent read %d bytes : %s\n", nbr, arr);
    }
    return 0;
}
```

Result:

```
Enter a string: Hello World
Child wrote 11 bytes
Parent read 11 bytes : Hello World
```

Program 6

Objective

Implementation of FIFO

Solution

FIFO (Writer Process)

Code:

```c
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>

int main()
{
    int fd, nbw;
    char str[100];
    mknod("myfifo", S_IFIFO | 0666, 0);
    printf("Writing for reader Process:\n\t");
    fd = open("myfifo", O_WRONLY);
    while (gets(str))
    {
        nbw = write(fd, str, strlen(str));
        printf("Writer process write %d bytes: %s\n", nbw, str);
    }
    return 0;
}
```

Result:

```
Writing for reader Process: computers
Writer process write 9 bytes: computers
```

FIFO (Reader Process)

Code:

```c
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>

int main()
{
    int fd, nbr;
    char arr[100];
    mknod("myfifo", S_IFIFO | 0666, 0);
```

```
        fd = open("myfifo", O_RDONLY);
        printf("If you got a writer process then type some data \n");
        do
        {
            nbr = read(fd, arr, sizeof(arr));
            arr[nbr] = '\0';
            printf("Reader process read %d bytes: %s\n", nbr, arr);
        } while (nbr > 0);
        return 0;
    }
```

Result:

```
If you got a writer process then type some data
Reader process read 9 bytes: computers
```

Program 7

Objective

Implementation of Shared Memory

Solution

Shared Memory (Writer Process)

Code:

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

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

int main()
{
    key_t key;
    int msgid, c;
    key = ftok("progfile", 'A');
    msgid = msgget(key, 0666 | IPC_CREAT);
    svarname.mtype = 1;
    printf("\nEnter a string : ");
    gets(svarname.mtext);
    c = msgsnd(msgid, &svarname, strlen(svarname.mtext), 0);
    printf("Sender wrote the text :\t %s \n", svarname.mtext);
    return (0);
}
```

Result:

```
Enter a string : Hello, World
Sender wrote the text :  Hello, World
```

Shared Memory (Reader Process)

Code:

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

```c
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>

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

int main()
{
    key_t key;
    int msgid, c;
    key = ftok("progfile", 'A');
    msgid = msgget(key, 0666 | IPC_CREAT);
    msgrcv(msgid, &svarname, sizeof(svarname), 1, 0);
    printf("Data Received is : \t %s \n", svarname.mtext);
    msgctl(msgid, IPC_RMID, NULL);
    return 0;
}
```

Result:

```
Data Received is :      Hello, World
```

Program 8

Objective

Implementation of Shared Memory

Solution

Shared Memory (Writer Process)

Code:

```c
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>

int main()
{
    key_t key;
    int shmid;
    void *ptr;
    key = ftok("shmfile", 'A');
    shmid = shmget(key, 1024, 0666 | IPC_CREAT);
    ptr = shmat(shmid, (void *)0, 0);
    printf("\nInput Data : ");
    gets(ptr);
    shmdt(ptr);
    return 0;
}
```

Result:

```
Input Data : Hello, World
```

Shared Memory (Reader Process)

Code:

```c
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>

int main()
{
```

```
    key_t key;
    int shmid;
    void *ptr;
    key = ftok("srfile", 'A');
    shmid = shmget(key, 1024, 0666 | IPC_CREAT);
    ptr = shmat(shmid, (void *)0, 0);
    printf("\nThe Data stored : %s\n", ptr);
    shmdt(ptr);
    shmctl(shmid, IPC_RMID, NULL);
    return (0);
}
```

Result:

```
The Data stored : Hello, World
```

Shared Memory (Reader and Writer Process Together)

Code:

```
#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>

int main()
{
    key_t key;
    int shmid;
    void *ptr;
    key = ftok("srfile", 'A');
    shmid = shmget(key, 1024, 0666 | IPC_CREAT);
    ptr = shmat(shmid, (void *)0, 0);
    printf("Input Data: ");
    gets(ptr);
    printf("The Data stored: %s\n", ptr);
    shmdt(ptr);
    shmctl(shmid, IPC_RMID, NULL);
    return (0);
}
```

Result:

```
Input Data:Hello, World!
The Data stored: Hello, World!
```

Program 9

Objective

Implementation of FCFS (First Come First Serve)

Solution

Using Pointers

Code:

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

typedef struct node
{
    char prss[3];
    int burst;
    int arrival;
    struct node *next;
} node;

node *front = NULL;
node *rear = NULL;

void insert();

void display(int);

void main()
{
    int i, n;
    printf("\nEnter number of processes : ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        insert();
        printf("\n\nExecuting processes : \n");
        display(n);
        printf("\n");
    }
}
```

```
void insert()
{
    node *p;
    int b, a;
    char str[3];
    p = (node *)malloc(sizeof(node));
    printf("\n\tEnter the process name : ");
    scanf("%s", p->prss);
    printf("\tEnter Burst time : ");
    scanf("%d", &b);
    printf("\tEnter arrival time : ");
    scanf("%d", &a);
    p->burst = b;
    p->arrival = a;
    p->next = NULL;
    if (front == NULL)
    {
        front = p;
        rear = p;
    }
    else
    {
        rear->next = p;
        rear = p;
    }
}

void display(int n)
{
    node *temp = front;
    int wttime = 0, c = 0;
    float turn = 0.0;
    if (front != NULL)
    {
        printf("\n----------\n\t");
        while (temp != NULL)
        {
            printf("|\t%s\t", temp->prss);
            temp = temp->next;
        }
        printf("|\n----------\n\t");
        temp = front;
        while (temp != NULL)
        {
            printf(" \t%d\t ", temp->burst);
            temp = temp->next;
        }
        printf("\n----------\n\t");
```

```
            temp = front;
            printf("0\t");
            while (temp != NULL)
            {
                wttime += c;
                turn += c + temp->burst;
                c = c + temp->burst;
                printf(" \t%d\t ", c);
                temp = temp->next;
            }
            printf("\n----------------------\n");
            printf("\n\nAveragewt time = %d ", wttime / n);
            printf("\nTurnaround time = %f\n", turn / n);
        }
    }
```

Result:

```
Enter number of processes : 3
Enter the process 1 name : P1
Enter Burst time : 24
Enter arrival time : 0
Enter the process 2 name : P2
Enter Burst time : 3
Enter arrival time : 0
Enter the process 3 name : P3
Enter Burst time : 3
Enter arrival time : 0
Executing processes :
------------------------------------------------------------------ | P1 |
P2 | P3 |
------------------------------------------------------------------ 24 3 3
------------------------------------------------------------------ 0 24 27
30
Average wt time = 17 Turnaround time = 27.000000
```

Using Arrays

Code:

```
#include <stdio.h>
#include <process.h>

void main()
{
    char p[10][5], temp[5];
    int c = 0, pt[10], i, j, n, temp1;
    float bst = 0.0, turn = 0.0;
    clrscr();
    printf("enter no of processes:");
    scanf("%d", &n);
```

```
    for (i = 0; i < n; i++)
    {
        printf("enter process%d name:\n", i + 1);
        scanf("%s", &p[i]);
        printf("enter process time");
        scanf("%d", &pt[i]);
    }
    printf("\n.....................................................
\n");
    for (i = 0; i < n; i++)
    {
        printf("|\t %s\t", p[i]);
    }
    printf("|
\n.....................................................\n");
    for (i = 0; i < n; i++)
    {
        printf("\t\t%d", pt[i]);
    }
    printf("\n.....................................................
\n");
    printf("0");
    for (i = 0; i < n; i++)
    {
        bst += c;
        turn += c + pt[i];
        c = c + pt[i];
        printf("\t\t%d", c);
    }
    printf("\nAverage time is %f: ", bst / n);
    printf("\nTurn around time is %f", turn / n);
}
```

Result:

```
Enter number of processes : 3
Enter the process 1 name : P1
Enter Burst time : 24
Enter arrival time : 0
Enter the process 2 name : P2
Enter Burst time : 3
Enter arrival time : 0
Enter the process 3 name : P3
Enter Burst time : 3
Enter arrival time : 0
Executing processes :
_____________________________________________________ | P1 |
P2 | P3 |
_____________________________________________________ 24 3 3
```

```
———————————————————————————————————————— 0 24 27
30
Average wt time = 17 Turnaround time = 27.000000
```

Program 10

Objective

Implementation of SJFS (Shortest Job First Scheduling)

Solution

Using Pointers

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct node
{
    char prss[3];
    int burst;
    struct node *next;
} node;

node *front = NULL;
node *rear = NULL;

void insert();

void display(int);

void main()
{
    int i, n;
    printf("\nEnter number of processes : ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        insert();
    printf("\n\nExecuting processes : \n");
    display(n);
    printf("\n");
}

void insert()
{
    node *p, *temp;
```

```
    int b;
    p = (node *)malloc(sizeof(node));
    printf("\n\tEnter the process name : ");
    scanf("%s", p->prss);
    printf("\tEnter Burst time : ");
    scanf("%d", &b);
    p->burst = b;
    p->next = NULL;
    if (front == NULL)
    {
        front = p;
        rear = p;
    }
    else if (p->burst < front->burst)
    {
        p->next = front;
        front = p;
    }
    else if (p->burst > rear->burst)
    {
        rear->next = p;
        rear = p;
    }
    else
    {
        temp = temp->next;
        p->next = temp->next;
        temp->next = p;
    }
}

void display(int n)
{
    node *temp = front;
    int c = 0;
    float turn = 0.0, wttime = 0.0;
    if (front != NULL)
    {
        printf("\n-------------------------\n\t");
        while (temp != NULL)
        {
            printf("|\t%s\t", temp->prss);
            temp = temp->next;
        }
        printf("\n-------------------------\n\t");
        temp = front;
        while (temp != NULL)
        {
```

```
            printf(" \t%d\t ", temp->burst);
            temp = temp->next;
        }
        printf("\n--------------------------\n\t");
        temp = front;
        printf("0\t");
        while (temp != NULL)
        {
            wttime += c;
            turn += c + temp->burst;
            c = c + temp->burst;
            printf(" \t%d\t ", c);
            temp = temp->next;
        }
        printf("\n--------------------------\n");
        printf("\n\nAveragewt time = %f ", wttime / n);
        printf("\nTurn around time = %f\n", turn / n);
    }
}
```

Result:

```
Enter number of processes : 3
Enter the process 1 name : P1
Enter Burst time : 24
Enter the process 2 name : P2
Enter Burst time : 2
Enter the process 3 name : P3
Enter Burst time : 3
Executing processes :
------------------------------------------------------------------------ | P2 |
P3 | P1 |
-------------------------------------------------------------------------
2 3 24 ------------------------------------------------------------------------
0 2 5 29
Average wt time = 2.333333 Turnaround time = 12.000000
```

Using Arrays

Code:

```
#include <stdio.h>
#include <process.h>

void main()
{
    char p[10][5], temp[5];
    int c = 0, pt[10], i, j, n, temp1;
    float bst = 0.0, turn = 0.0;
    clrscr();
    printf("enter no of processes:");
```

```
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("enter process%d name:\n", i + 1);
        scanf("%s", &p[i]);
        printf("enter process time");
        scanf("%d", &pt[i]);
    }
    for (i = 0; i < n - 1; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (pt[i] > pt[j])
            {
                temp1 = pt[i];
                pt[i] = pt[j];
                pt[j] = temp1;
                strcpy(temp, p[i]);
                strcpy(p[i], p[j]);
                strcpy(p[j], temp);
            }
        }
    }
    printf("\n.....
\n");
    for (i = 0; i < n; i++)
    {
        printf("|\t %s\t", p[i]);
    }
    printf("|
\n.....\n");
    for (i = 0; i < n; i++)
    {
        printf("\t\t%d", pt[i]);
    }
    printf("\n.....
\n");
    printf("0");
    for (i = 0; i < n; i++)
    {
        bst += c;
        turn += c + pt[i];
        c = c + pt[i];
        printf("\t\t%d", c);
    }
    printf("\nAverage time is %f: ", bst / n);
    printf("\nTurn around time is %f ", turn / n);
}
```

Result:

```
Enter number of processes : 3
Enter the process 1 name : P1
Enter Burst time : 24
Enter the process 2 name : P2
Enter Burst time : 2
Enter the process 3 name : P3
Enter Burst time : 3
Executing processes :
_____________________________________________________________ | P2 |
P3 | P1 |
_____________________________________________________________
2 3 24 ____________________________________________________________
0 2 5 29
Average wt time = 2.333333 Turnaround time = 12.000000
```

Program 11

Objective

Implementation of Priority Scheduling

Solution

Using Pointers

Code:

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

typedef struct node
{
    charprss[3];
    int burst;
    int priority;
    struct node *next;
} node;

node *front = NULL;
node *rear = NULL;

void insert();

void display(int);

void main()
{
    int i, n;
    printf("\nEnter number of processes : ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        insert();
    printf("\n\nExecuting processes : \n");
    display(n);
    printf("\n");
}

void insert()
{
    node *p, *temp;
```

```
    int b, pri;
    p = (node *)malloc(sizeof(node));
    printf("\n\tEnter the process name : ");
    scanf("%s", p->prss);
    printf("\tEnter Burst time : ");
    scanf("%d", &b);
    printf("\tEnter Priority : ");
    scanf("%d", &pri);
    p->burst = b;
    p->priority = pri;
    p->next = NULL;
    if (front == NULL)
    {
        front = p;
        rear = p;
    }
    else if (p->priority < front->priority)
    {
        p->next = front;
        front = p;
    }
    else if (p->priority > rear->priority)
    {
        rear->next = p;
        rear = p;
    }
    else
    {
        temp = front;
        while (p->priority > (temp->next)->priority)
            temp = temp->next;
        p->next = temp->next;
        temp->next = p;
    }
}

void display(int n)
{
    node *temp = front;
    int c = 0;
    float turn = 0.0, wttime = 0.0;
    if (front != NULL)
    {
        printf("\n----------------------\n\t");
        while (temp != NULL)
        {
            printf("|\t%s\t", temp->prss);
            temp = temp->next;
```

```
            }
            printf("\n————————————————————\n");
            temp = front;
            while (temp ≠ NULL)
            {
                printf("\t%d\t ", temp→burst);
                temp = temp→next;
            }
            printf("\n————————————————————\n\t");
            temp = front;
            printf("0\t");
            while (temp ≠ NULL)
            {
                wttime += c;
                turn += c + temp→burst;
                c = c + temp→burst;
                printf(" \t%d\t ", c);
                temp = temp→next;
            }
            printf("\n————————————————————\n");
            printf("\n\nAveragewt time = %f ", wttime / n);
            printf("\nTurn around time = %f\n", turn / n);
        }
    }
```

Result:

```
Enter number of processes : 3
Enter the process name : P1
Enter Burst time : 24
Enter Priority : 3
Enter the process name : P2
Enter Burst time : 3
Enter Priority : 1
Enter the process name : P3
Enter Burst time : 2
Enter Priority : 2

Executing processes :
———————————————————————————————————————————————————————————————————————————— |
P2 | P3 | P1 |
————————————————————————————————————————————————————————————————————————————

3 2 24
————————————————————————————————————————————————————————————————————————————

0 3 5 29
Average wt time = 2.666667 Turnaround time = 12.333333
```

Using Arrays

Code:

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

void main()
{
    char p[10][5], temp[5];
    int c = 0, pt[10], pr[i], i, j, n, temp1;
    float bst = 0.0, turn = 0.0;
    clrscr();
    printf("enter no of processes:");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("enter process%d name:\n", i + 1);
        scanf("%s", &p[i]);
        printf("enter process time");
        scanf("%d", &pt[i]);
        printf("\nenter the priority of process");
        scanf(“% d”, &pr[i]);
    }
    for (i = 0; i < n - 1; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (pr[i] > pr[j])
            {
                temp1 = pt[i];
                pt[i] = pt[j];
                pt[j] = temp1;
                t = pr[i];
                pr[i] = pr[j];
                pr[j] = t;
                strcpy(temp, p[i]);
                strcpy(p[i], p[j]);
                strcpy(p[j], temp);
            }
        }
    }
    printf("\n.....
\n");
    for (i = 0; i < n; i++)
    {
        printf("|\t %s\t", p[i]);
    }
    printf("|
\n.....\n");
    for (i = 0; i < n; i++)
    {
```

```
            printf("\t\t%d", pt[i]);
        }
        printf("\n.....................................................
    \n");
        printf("0");
        for (i = 0; i < n; i++)
        {
            bst += c;
            turn += c + pt[i];
            c = c + pt[i];
            printf("\t\t%d", c);
        }
        printf("\nAverage time is %f: ", bst / n);
        printf("\nTurn around time is %f", turn / n);
    }
```

Result:

```
Enter number of processes : 3
Enter the process name : P1
Enter Burst time : 24
Enter Priority : 3
Enter the process name : P2
Enter Burst time : 3
Enter Priority : 1
Enter the process name : P3
Enter Burst time : 2
Enter Priority : 2

Executing processes :
_____________________________________________________________________ |
P2 | P3 | P1 |
_____________________________________________________________________
3 2 24
_____________________________________________________________________
0 3 5 29
Average wt time = 2.666667 Turnaround time = 12.333333
```