SLIP-1

1 Write a program that demonstrates the use of nice() system call. After a child process is started using fork(), assign higher priority to the child using nice() system call.

ANSWER-1

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main() {

    pid_t child_pid;

    child_pid = fork();

    if (child_pid == 0) {

        printf("Child process with PID: %d\n", getpid());

        nice(10);

        printf("Child process priority increased\n");

    } else if (child_pid > 0) {

        printf("Parent process with PID: %d\n", getpid());

        wait(NULL);

    } else {

        perror("Fork failed");

        return 1;}

    return 0;}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n=3 as the number of memory frames. Reference String :3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6 Implement FIFO

ANSWER-2

```
#include <stdio.h>

int main() {

    int n = 3, memory[3] = {-1}, faults = 0;

    int pages[] = {3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6};

    for (int i = 0; i < 15; i++) {

        int p = pages[i], found = 0;

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

            if (memory[j] == p) {

                found = 1;

                break;}}

        if (!found) {

            faults++;

            memory[faults % n] = p;}}

    printf("Total Page Faults: %d\n", faults);

    return 0;}
```

SLIP-2

Q.1 Create a child process using fork(), display parent and child process id. Child process will display the message “Hello World” and the parent process should display “Hi”.

ANSWER-1

```
#include <stdio.h>

#include <unistd.h>

int main() {
    pid_t child_pid = fork();
    if (child_pid == 0) {
        printf("Hello World\n");
    } else if (child_pid > 0) {
        printf("Hi\n");
    } else {
        perror("Fork failed");
        return 1;}
    return 0;}
```

Write the simulation program using SJF (non-preemptive). The arrival time and first CPU bursts of different jobs should be input to the system. Assume the fixed I/O waiting time (2 units). The next CPU burst should be generated using random function. The output should give the Gantt chart, Turnaround Time and Waiting time for each process and average times

ANSWER-2

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX_PROCESSES 10
typedef struct {
    int arrival, burst, turnaround, waiting;
} Process;
int cmpArrival(const void *a, const void *b) { return ((Process *)a)->arrival - ((Process *)b)->arrival; }
int main() {
    int n; scanf("%d", &n);
    Process p[MAX_PROCESSES] = {0};
    srand(time(0));
    for (int i = 0; i < n; i++) p[i].arrival = i, p[i].burst = rand() % 10 + 1;
    qsort(p, n, sizeof(Process), cmpArrival);
    printf("Gantt Chart:\n");
    float tTAT = 0, tWT = 0, t = 0;
    for (int i = 0; i < n; i++) {
        printf("P%d ", i + 1);
        p[i].turnaround = t + p[i].burst - p[i].arrival;
        p[i].waiting = p[i].turnaround - p[i].burst;
        t += p[i].burst + 2;
        tTAT += p[i].turnaround;
        tWT += p[i].waiting;}
    printf("\nProcess\tTurnaround\tWaiting\n");
    for (int i = 0; i < n; i++) printf("P%d\t%d\t\t%d\n", i + 1, p[i].turnaround, p[i].waiting);
    printf("Avg TAT: %.2f\nAvg WT: %.2f\n", tTAT / n, tWT / n);}
```

SLIP-3

Q. 1 Creating a child process using the command exec(). Note down process ids of the parent and the child processes, check whether the control is given back to the parent after the child process terminates.

ANSWER-1

```
#include <stdio.h>
#include <unistd.h>
int main() {
    pid_t child_pid = fork();
    if (child_pid == 0) {
        printf("Child process (PID: %d)\n", getpid());
        execl("/bin/ls", "ls", NULL); // Replace with the desired command
    } else if (child_pid > 0) {
        printf("Parent process (PID: %d)\n", getpid());
        wait(NULL);
        printf("Child process has terminated.\n");
    } else {
        perror("Fork failed");
        return 1;}
    return 0;}
```

Q.2 Write the simulation program using FCFS. The arrival time and first CPU bursts of different jobs should be input to the system. Assume the fixed I/O waiting time (2 units). The next CPU burst should be generated using random function. The output should give the Gantt chart,Turnaround Time and Waiting time for each process and average times.

ANSWER-2

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX_PROCESSES 10
typedef struct {
    int arrival, burst, turnaround, waiting;
} Process;
int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    Process p[MAX_PROCESSES] = {0};
    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        p[i].arrival = i;
        p[i].burst = rand() % 10 + 1; }
    printf("Gantt Chart:\n");
    float tTAT = 0, tWT = 0, t = 0;
    for (int i = 0; i < n; i++) {
        printf("P%d ", i + 1);
        t += p[i].burst + 2; // Add I/O waiting time
        p[i].turnaround = t - p[i].arrival;
        p[i].waiting = p[i].turnaround - p[i].burst;
        tTAT += p[i].turnaround;
        tWT += p[i].waiting;}
    printf("\n");
    printf("Process\tTurnaround\tWaiting\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t\t%d\n", i + 1, p[i].turnaround, p[i].waiting);}
    printf("Avg TAT: %.2f\nAvg WT: %.2f\n", tTAT / n, tWT / n);
    return 0;}
```

SLIP-4

Write a program to illustrate the concept of orphan process (Using fork() and sleep())

ANSWER-1

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main() {
    pid_t child_pid = fork();
    if (child_pid == 0) {
        printf("Child process (PID: %d) is running, and its parent (PPID: %d) has terminated.\n", getpid(), getppid());
        sleep(5);
        printf("Child process (PID: %d) has completed.\n", getpid());
    } else if (child_pid > 0) {
        printf("Parent process (PID: %d) is running and will terminate shortly.\n", getpid());
        sleep(2);
        printf("Parent process (PID: %d) has terminated.\n", getpid());
    } else {
        perror("Fork failed");
        return 1;}
    return 0;}
```

Q.2 Write the program to simulate Non-preemptive Priority scheduling. The arrival time and first CPU burst and priority for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

ANSWER-2

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 10
typedef struct { int p, b, tT, tW; } P;
int main() {
    int n, t = 0, tTAT = 0, tWT = 0;
    srand(time(0));
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    P p[N] = {0};
    for (int i = 0; i < n; i++) p[i].p = rand() % 10, p[i].b = rand() % 10 + 1;
    printf("Gantt Chart:\n");
    for (int i = 0; i < n; i++) {
        int m = -1;
        for (int j = 0; j < n; j++) if (p[j].p < p[i].p) m = j;
        printf("P%d ", m + 1);
        p[m].tT = t + p[m].b, p[m].tW = t, t += p[m].b + 2;
        tTAT += p[m].tT, tWT += p[m].tW, p[m].p = 11;}
    printf("\nProcess\tTurnaround\tWaiting\n");
    for (int i = 0; i < n; i++) printf("P%d\t%d\t\t%d\n", i + 1, p[i].tT, p[i].tW);
    printf("Avg TAT: %.2f\nAvg WT: %.2f\n", (float)tTAT / n, (float)tWT / n);
    return 0;}
```

SLIP-5

Q.1 Write a program that demonstrates the use of nice () system call. After a child process is started using fork (), assign higher priority to the child using nice () system call.

ANSWER-1

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main() {
    pid_t child_pid = fork();
    if (child_pid == 0) {
        printf("Child process with PID: %d\n", getpid());
        nice(10);
        printf("Child process priority increased\n");
    } else if (child_pid > 0) {
        printf("Parent process with PID: %d\n", getpid());
        wait(NULL);
    } else {
        perror("Fork failed");
        return 1;}
    return 0;}
```

2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames. Reference String: 3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6 i. Implement FIFO

ANSWER-2

```
#include <stdio.h>
int main() {
    int n = 3, referenceString[] = {3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6};
    int memory[n], faults = 0, i = 0;
    for (int j = 0; j < n; j++) memory[j] = -1;
    printf("Page Scheduling using FIFO:\n");
    for (int r = 0; r < 15; r++) {
        int page = referenceString[r], found = 0;
        for (int j = 0; j < n; j++) {
            if (memory[j] == page) {
                found = 1;
                break;}}
        if (!found) {
            faults++;
            if (memory[i % n] != -1) {
                printf("Page %d replaced by Page %d\n", memory[i % n], page);}
            memory[i % n] = page;
            i++;
            printf("Page %d -> Page Fault\nMemory: ", page);
            for (int j = 0; j < n; j++) {
                printf("%d ", memory[j]);}
            printf("\n");}}
    printf("Total Page Faults: %d\n", faults);
    return 0;}
```

SLIP-6

Q.1 Write a program to find the execution time taken for execution of a given set of instructions (use clock() function

ANSWER-1

```
 #include <stdio.h>
#include <time.h>
int main() {
    clock_t start = clock();
    clock_t end = clock();
    double executionTime = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Execution time: %.4f seconds\n", executionTime);
    return 0;}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames. Reference String :3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6 Implement FIFO

ANSWER-2

```
#include <stdio.h>
int main() {
    int n; printf("Enter memory frames: "); scanf("%d", &n);
    int ref[] = {3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6};
    int len = sizeof(ref) / sizeof(ref[0]), mem[n], faults = 0, i = 0;
    for (int j = 0; j < n; j++) mem[j] = -1;
    printf("Page Scheduling using FIFO:\n");
    for (int r = 0; r < len; r++) {
        int p = ref[r], found = 0;
        for (int j = 0; j < n; j++) {
            if (mem[j] == p) {
                found = 1;
                break;}}
        if (!found) {
            faults++;
            if (mem[i % n] != -1) {
                printf("Page %d replaced by Page %d\n", mem[i % n], p);}
            mem[i % n] = p;
            i++;
            printf("Page %d -> Page Fault\nMemory: ", p);
            for (int j = 0; j < n; j++) {
                printf("%d ", mem[j]);}
            printf("\n");}}
    printf("Total Page Faults: %d\n", faults);
    return 0;}
```

SLIP-7

Q.1 Write a program to create a child process using fork().The parent should goto sleep state and child process should begin its execution. In the child process, use execl() to execute the “ls” command.

ANSWER-1

```
#include <stdio.h>

#include <unistd.h>

int main() {

    pid_t child_pid = fork();

    if (child_pid == 0) {

        execl("/bin/ls", "ls", (char *)NULL);

    } else if (child_pid > 0) {

        sleep(5);

    } else {

        perror("Fork failed");

        return 1;}

    return 0;}
```

Q.2 Write the simulation program using FCFS. The arrival time and first CPU bursts of different jobs should be input to the system. Assume the fixed I/O waiting time (2 units). The next CPU burst should be generated using random function. The output should give the Gantt chart, Turnaround Time and Waiting time for each process and average times

ANSWER-2

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main() {

    int n; srand(time(NULL));

    printf("Enter the number of processes: "); scanf("%d", &n);

    float tTAT = 0, tWT = 0, t = 0;

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

        int burst = rand() % 10 + 1;

        t += burst + 2;

        printf("P%d %d %d\n", i + 1, (int)t, (int)t - burst);

        tTAT += t;

        tWT += t - burst;}

    printf("Avg TAT: %.2f\nAvg WT: %.2f\n", tTAT / n, tWT / n);

    return 0;

}
```

SLIP-8

Q.1 Write a C program to accept the number of process and resources and find the need matrix content and display it.

ANSWER-1

```
#include <stdio.h>

int main() {

    int n, m;

    printf("Enter processes and resources: ");

    scanf("%d %d", &n, &m);
    int alloc[n][m], max[n][m], need[n][m];

    printf("Enter allocation and max matrices:\n");

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

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

            scanf("%d %d", &alloc[i][j], &max[i][j]);

            need[i][j] = max[i][j] - alloc[i][j];}}

    printf("Need matrix:\n");

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

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

            printf("%d ", need[i][j]);}

        printf("\n");}

    return 0;}
```

Q.2. Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n =3 as the number of memory frames. Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8 Implement OPT

ANSWER-2

```
#include <stdio.h>

int main() {

    int n; printf("Enter memory frames: "); scanf("%d", &n);

    int ref[] = {12, 15, 12, 18, 6, 8, 11, 12, 19, 12, 6, 8, 12, 15, 19, 8};

    int len = sizeof(ref) / sizeof(ref[0]), mem[n], faults = 0;

    for (int j = 0; j < n; j++) mem[j] = -1;

    printf("Page Scheduling using OPT:\n");

    for (int r = 0; r < len; r++) {

        int p = ref[r], found = 0, future = -1, fIdx = 0;

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

            if (mem[j] == p) {

                found = 1;

                break;}

            for (int k = r + 1; k < len; k++) {

                if (mem[j] == ref[k]) {

                    future = k;

                    break;}}

            if (future == -1) {

                fIdx = j;

                break;}

            if (future > fIdx) {

                fIdx = j;}}

        if (!found) {

            faults++;

            if (mem[fIdx] != -1) {

                printf("Page %d replaced by Page %d\n", mem[fIdx], p);}

            mem[fIdx] = p;

            printf("Page %d -> Page Fault\nMemory: ", p);

            for (int j = 0; j < n; j++) {
                printf("%d ", mem[j]);}

            printf("\n");}}

    printf("Total Page Faults: %d\n", faults);

    return 0;}
```

SLIP-9

Q.1 Write a program to create a child process using fork().The parent should goto sleep state and child process should begin its execution. In the child process, use execl() to execute the “ls” command.

ANSWER-1

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main() {

    pid_t child_pid = fork();

    if (child_pid == 0) {

        execl("/bin/ls", "ls", (char *)NULL);

        perror("execl() failed");

    } else if (child_pid > 0) {

        sleep(5);

    } else {

        perror("Fork failed");

        return 1;}

    return 0;}
```

Q.2 Write the program to simulate Round Robin (RR) scheduling. The arrival time and first CPUburst for different n number of processes should be input to the algorithm. Also give the time quantum as input. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

ANSWER-2

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

typedef struct { int a, b, r, t, w; } P;

int main() {

    int n, q, s = 0, i = 0; srand(time(0));

    printf("Enter n and quantum: "); scanf("%d%d", &n, &q);

    P p[n]; for (int j = 0; j < n; j++) { p[j].a = j; p[j].b = rand() % 10 + 1; p[j].r = p[j].b; }

    printf("Gantt Chart:\n");

    while (1) {

        int d = 1;

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

            if (p[j].r > 0) {

                d = 0;

                int u = p[j].r > q ? q : p[j].r;

                p[j].r -= u; s += u;

                printf("P%d ", j + 1);

                p[j].a = s; p[j].r = 0;}}

        if (d) break;}
    printf("\nTurnaround\tWaiting\n");
    for (int j = 0; j < n; j++) {
        p[j].t = p[j].a - j;
        p[j].w = p[j].t - p[j].b;
        printf("P%d\t%d\t\t%d\n", j + 1, p[j].t, p[j].w);
        i += p[j].t; q += p[j].w;}
    printf("Avg TAT: %.2f\nAvg WT: %.2f\n", (float)i / n, (float)q / n);
    return 0;}
```

SLIP-10

Q.1 Write a program to illustrate the concept of orphan process (Using fork() and sleep())

ANSWER-1

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main() {
    pid_t child_pid = fork();
    if (child_pid == 0) {
        printf("Child process (PID %d) is running.\n", getpid());
        sleep(2);
        printf("Child process (PID %d) is done.\n", getpid());
    } else if (child_pid > 0) {
        printf("Parent process (PID %d) is running.\n", getpid());
        printf("Parent process (PID %d) is done.\n", getpid());
    } else {
        perror("Fork failed");
        return 1;}
    return 0;}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n=3 as the number of memory frames. Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8 Implement OPT

ANSWER-2

```
#include <stdio.h>
int main() {
    int n, ref[] = {12, 15, 12, 18, 6, 8, 11, 12, 19, 12, 6, 8, 12, 15, 19, 8};
    int len = sizeof(ref) / sizeof(ref[0]), mem[n], faults = 0, future[len];
    printf("Enter number of memory frames: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) mem[i] = -1;
    for (int i = 0; i < len; i++) {
        int p = ref[i], found = 0;
        for (int j = 0; j < n; j++) {
            if (mem[j] == p) {
                found = 1;
                break;}
            future[j] = len;
            for (int k = i + 1; k < len; k++) {
                if (mem[j] == ref[k]) {
                    future[j] = k;
                    break;}}}
        if (!found) {
            faults++;
            int fIdx = 0;
            for (int j = 0; j < n; j++) {
                if (future[j] > future[fIdx]) {
                    fIdx = j;}}
            mem[fIdx] = p;}}
    printf("Total Page Faults: %d\n", faults);
    return 0;}
```

SLIP-11

Q.1 Create a child process using fork(), display parent and child process id. Child process will display the message “Hello World” and the parent process should display “Hi”.

ANSWER-1

```
#include <stdio.h>
#include <unistd.h>
int main() {
    pid_t child_pid = fork();
    if (child_pid == 0) {
        printf("Child Process ID: %d\n", getpid());
        printf("Hello World\n");
    } else if (child_pid > 0) {
        printf("Parent Process ID: %d\n", getpid());
        printf("Hi\n");
    } else {
        perror("Fork failed");
        return 1;}
    return 0;}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames. Reference String: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1 Implement FIFO

ANSWER-2

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int n;
    printf("Enter the number of memory frames: ");
    scanf("%d", &n);
    int referenceString[] = {0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1};
    int referenceStringLength = sizeof(referenceString) / sizeof(referenceString[0]);
    int memory[n];
    int pageFaults = 0;
    int nextToReplace = 0;
    for (int i = 0; i < n; i++) {
        memory[i] = -1;}
    printf("Page Scheduling using FIFO:\n");
    for (int i = 0; i < referenceStringLength; i++) {
        int page = referenceString[i];
        int found = 0;
        for (int j = 0; j < n; j++) {
            if (memory[j] == page) {
                found = 1;
                break;}}
        if (!found) {
            pageFaults++;
            if (memory[nextToReplace] != -1) {
                printf("Page %d replaced by Page %d\n", memory[nextToReplace], page);}
            memory[nextToReplace] = page;
            nextToReplace = (nextToReplace + 1) % n;
            printf("Page %d -> Page Fault\n", page);}}
    printf("Total Page Faults: %d\n", pageFaults);
    return 0;}
```

SLIP-12

Q.1 Create a child process using fork(), display parent and child process id. Child process will display the message “Hello World” and the parent process should display “Hi”.

ANSWER-1

```
#include <stdio.h>
#include <unistd.h>
int main() {
    pid_t child_pid = fork();
    if (child_pid == 0) {
        printf("Child Process ID: %d\n", getpid());
        printf("Hello World\n");
    } else if (child_pid > 0) {
        printf("Parent Process ID: %d\n", getpid());
        printf("Hi\n");
    } else {
        perror("Fork failed");
        return 1;}
    return 0;}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames. Reference String: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1 Implement FIFO

ANSWER-2

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int n;
    printf("Enter the number of memory frames: ");

    scanf("%d", &n);

    int referenceString[] = {0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1};

    int referenceStringLength = sizeof(referenceString) / sizeof(referenceString[0]);

    int memory[n];

    int pageFaults = 0;

    int replaceIndex = 0;

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

        memory[i] = -1;}

    printf("Page Scheduling using FIFO:\n");

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

        int page = referenceString[i];

        int found = 0;

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

            if (memory[j] == page) {

                found = 1;

                break;}}

        if (!found) {

            pageFaults++;

            memory[replaceIndex] = page;

            replaceIndex = (replaceIndex + 1) % n;

            printf("Page %d -> Page Fault\n", page);}}

    printf("Total Page Faults: %d\n", pageFaults);

    return 0;}
```

SLIP-13

Write a program to illustrate the concept of orphan process (Using fork() and sleep()) .

ANSWER-1

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main() {

    pid_t child_pid = fork();

    if (child_pid == 0) {

        printf("Child process (PID %d) is running.\n", getpid());

        sleep(100);

        printf("Child process (PID %d) is done.\n", getpid());

    } else if (child_pid > 0) {

        printf("Parent process (PID %d) is running.\n", getpid());

        printf("Parent process (PID %d) is done.\n", getpid());

    } else {

        perror("Fork failed");

        return 1;}

    return 0;}
```

Write the simulation program using SJF(non-preemptive). The arrival time and first CPU bursts of different jobsshould be input to the system. The Assume the fixed I/O waiting time (2 units).Thenext CPU burst should be generated using random function. The output should give the Gantt chart, Turnaround Time and Waiting time for each process and average times.

ANSWER-2

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

typedef struct { int a, b, t, w; } P;

int main() {

    int n, io = 2, t = 0; srand(time(0));

    printf("Enter n: "); scanf("%d", &n);

    P p[n]; for (int i = 0; i < n; i++) {

        p[i].a = i; printf("Arrival time for P%d: ", i + 1);

        scanf("%d", &p[i].t); printf("Burst time for P%d: ", i + 1);

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

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

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

            if (p[i].t > p[j].t) {

                P temp = p[i]; p[i] = p[j]; p[j] = temp;}}}

    printf("Gantt Chart:\n");

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

        printf("P%d ", p[i].a + 1);

        t += p[i].w;

        p[i].w = t - p[i].t;

        p[i].t = p[i].w + p[i].w;}

    printf("\nTurnaround Time\tWaiting Time\n");

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

        printf("P%d\t%d\t\t%d\n", p[i].a + 1, p[i].t, p[i].w);}

    return 0;}
```

SLIP-14

Write a program to find the execution time taken for execution of a given set of instructions (use clock() function)

ANSWER-1

```
#include <stdio.h>

#include <time.h>

int main() {

    clock_t start_time, end_time;

    double execution_time;

    start_time = clock();

    for (int i = 0; i < 1000000; i++) {

        int result = i * i;}

    end_time = clock();

    execution_time = (double)(end_time - start_time) / CLOCKS_PER_SEC;

    printf("Execution time: %f seconds\n", execution_time);

    return 0;}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n =3 as the number of memory frames. Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1 Implement FIFO

ANSWER-2

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int n, f = 0, p = 0, r, q = 0;
    printf("Enter the number of memory frames: ");
    scanf("%d", &n);
    int frames[n];
    printf("Reference String: ");
    while (scanf("%d", &r) != EOF) {
        int found = 0;
        for (int i = 0; i < n; i++) {
            if (frames[i] == r) {
                found = 1;
                break;}}
        if (!found) {
            printf("Page %d -> Page Fault\n", r);
            frames[f] = r;
            f = (f + 1) % n;
            p++;}
        q++;}
    printf("Total Page Faults: %d\n", p);
    return 0;}
```

SLIP-15

Write a program to create a child process using fork().The parent should goto sleep state and child process should begin its execution. In the child process, use execl() to execute the “ls” command.

ANSWER-1

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main() {
    pid_t child_pid = fork();
    if (child_pid < 0) {
        perror("Fork failed");
        exit(1);
    } else if (child_pid == 0) {
        printf("Child Process (PID %d) is running and executing 'ls'.\n", getpid());
        execl("/bin/ls", "ls", (char *)NULL);
        perror("execl() failed");
        exit(1);
    } else {
        printf("Parent Process (PID %d) is going to sleep.\n", getpid());
        sleep(2);
        printf("Parent Process (PID %d) is awake.\n", getpid());}
    return 0;}
```

Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames. Reference String :7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 Implement LRU

ANSWER-2

```
#include <stdio.h>

#include <stdlib.h>

int main() {

    int n, p = 0, r, q = 0;

    printf("Enter the number of memory frames: ");

    scanf("%d", &n);

    int frames[n];

    printf("Reference String: ");

    while (scanf("%d", &r) != EOF) {

        int found = 0;

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

            if (frames[i] == r) {

                found = 1;

                break;}}

        if (!found) {

            printf("Page %d -> Page Fault\n", r);

            frames[p % n] = r;

            p++;}

        q++;}

    printf("Total Page Faults: %d\n", p);

    return 0;}
```

SLIP-16

Q.1 Write a program to find the execution time taken for execution of a given set of instructions (use clock() function)

ANSWER-1

```
#include <stdio.h>

#include <time.h>

int main() {

    clock_t start_time, end_time;

    double execution_time;

    start_time = clock();

    for (int i = 0; i < 1000000; i++) {

        int result = i * i;}

    end_time = clock();

    execution_time = (double)(end_time - start_time) / CLOCKS_PER_SEC;

    printf("Execution time: %f seconds\n", execution_time);

    return 0;}
```

Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n =3 as the number of memory frames. Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8 Implement OPT

ANSWER-2

```
#include <stdio.h>

#include <stdlib.h>
```

```
int main() {
    int n, p = 0, r, q = 0;
    printf("Enter the number of memory frames: ");
    scanf("%d", &n);
    int frames[n];
    int nextUse[n];
    printf("Reference String: ");
    while (scanf("%d", &r) != EOF) {
        int found = 0;
        for (int i = 0; i < n; i++) {
            if (frames[i] == r) {
                found = 1;
                break;}}
        if (!found) {
            int toReplace = -1;
            for (int i = 0; i < n; i++) {
                nextUse[i] = q + 1;
                for (int j = q + 1; j < q + 1; j++) {
                    if (r == frames[i]) {
                        nextUse[i] = j;
                        break;}}}
            for (int i = 0; i < n; i++) {
                if (nextUse[i] == q + 1) {
                    toReplace = i;
                    break;}}
            if (toReplace == -1) {
                toReplace = 0;
                for (int i = 1; i < n; i++) {
                    if (nextUse[i] > nextUse[toReplace]) {
                        toReplace = i;}}}
            frames[toReplace] = r;
            p++;
            printf("Page %d -> Page Fault\n", r);}
        q++;}
    printf("Total Page Faults: %d\n", p);
    return 0;}
```

SLIP-17

Write the program to calculate minimum number of resources needed to avoid deadlock

ANSWER-1

```
#include <stdio.h>
int main() {
    int allocated[3][3] = {{1, 2, 3}, {2, 3, 4}, {3, 4, 5}};
    int max_demand[3][3] = {{4, 4, 4}, {5, 5, 5}, {6, 6, 6}};
    int available[3] = {0};
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            available[j] += max_demand[i][j] - allocated[i][j];}}
    printf("Minimum resources needed to avoid deadlock: ");
    for (int i = 0; i < 3; i++) {
        printf("%d ", available[i]);}
    printf("\n");
    return 0;}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n=3 as the number of memory frames. Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8 Implement OPT

ANSWER-2

```
#include <stdio.h>
int main() {
    int n, p = 0, r, q = 0;
    printf("Enter the number of memory frames: ");
    scanf("%d", &n);
    int frames[n];
    int nextUse[n];
    int referenceString[] = {12, 15, 12, 18, 6, 8, 11, 12, 19, 12, 6, 8, 12, 15, 19, 8};
    int referenceStringLength = sizeof(referenceString) / sizeof(referenceString[0]);
    for (int i = 0; i < n; i++) frames[i] = -1;
    printf("Page Scheduling using OPT:\n");
    while (q < referenceStringLength) {
        r = referenceString[q];
        int found = 0;
        for (int i = 0; i < n; i++) {
            if (frames[i] == r) {
                found = 1;
                break;}}
        if (!found) {
            int toReplace = -1;
            for (int i = 0; i < n; i++) {
                nextUse[i] = q + 1;
                for (int j = q + 1; j < referenceStringLength; j++) {
                    if (r == referenceString[j]) {
                        nextUse[i] = j;
                        break;}}}
            for (int i = 0; i < n; i++) {
                if (nextUse[i] == q + 1) {
                    toReplace = i;
                    break;}}
            if (toReplace == -1) {
                toReplace = 0;
                for (int i = 1; i < n; i++) {
                    if (nextUse[i] > nextUse[toReplace]) {
                        toReplace = i;}}}
            frames[toReplace] = r;
            p++;
            printf("Page %d -> Page Fault\n", r);}
        q++;}
    printf("Total Page Faults: %d\n", p);
    return 0;}
```

SLIP-18

Q. 1 Write a C program to accept the number of process and resources and find the need matrix content and display it.

ANSWER-1

```
#include <stdio.h>
int main() {
    int n, m;
    printf("Enter processes and resources: ");
    scanf("%d %d", &n, &m);
    int alloc[n][m], max[n][m], need[n][m];
    printf("Enter allocation and max matrices:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            scanf("%d %d", &alloc[i][j], &max[i][j]);
            need[i][j] = max[i][j] - alloc[i][j];}}
    printf("Need matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            printf("%d ", need[i][j]);}
        printf("\n");}
    return 0;}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames. Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8 Implement OPT

ANSWER-2

```
#include <stdio.h>
int main() {
    int n, faults = 0, i, j, page, farthest, index;
    printf("Enter frames: "); scanf("%d", &n);
    int ref[] = {12, 15, 12, 18, 6, 8, 11, 12, 19, 12, 6, 8, 12, 15, 19, 8};
    int frames[n];
    for (i = 0; i < n; i++) frames[i] = -1;
    for (i = 0; i < 16; i++) {
        page = ref[i];
        int found = 0;
        for (j = 0; j < n; j++) {
            if (frames[j] == page) {
                found = 1;
                break;}}
        if (!found) {
            faults++; farthest = -1;
            for (j = 0; j < n; j++) {
                int foundInFuture = 0;
                for (int k = i + 1; k < 16; k++) {
                    if (ref[k] == frames[j]) {
                        foundInFuture = 1;
                        if (k > farthest) {
                            farthest = k;
                            index = j;}
                        break;}}
                if (!foundInFuture) {
                    index = j;
                    break;}}
            frames[index] = page;
            printf("Page %d -> Page Fault\n", page);}}
    printf("Total Page Faults: %d\n", faults);
    return 0;}
```

SLIP-19

Q.1 Write a program to create a child process using fork().The parent should goto sleep state and child process should begin its execution. In the child process, use execl() to execute the “ls” command.

ANSWER-1

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main() {
    pid_t child_pid = fork();
    if (child_pid < 0) {
        perror("Fork failed");
        exit(1);}
    if (child_pid == 0) {
        execl("/bin/ls", "ls", NULL);
        perror("Execl failed");
        exit(1);
    } else {
        printf("Parent process is going to sleep for 3 seconds...\n");
        sleep(3);
        wait(NULL);
        printf("Parent process has woken up.\n");}
    return 0;}
```

Q.2 Write the program to simulate Non-preemptive Priority scheduling. The arrival time and first CPU burst and priority for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

ANSWER-2

```
#include <stdio.h>

struct Process {

    int pid;

    int arrival_time;

    int burst_time;

    int priority;};

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

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

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

            if (processes[i].priority > processes[j].priority) {

                struct Process temp = processes[i];

                processes[i] = processes[j];

                processes[j] = temp;}}}}

int main() {

    int n;

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

    scanf("%d", &n);

    struct Process processes[n];

    int total_turnaround_time = 0;

    int total_waiting_time = 0;

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

        processes[i].pid = i + 1;

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

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

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

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

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

        scanf("%d", &processes[i].priority);}

    sort_by_priority(processes, n);

    printf("\nGantt Chart:\n");

    int current_time = 0;

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

        printf("P%d -> ", processes[i].pid);

        total_waiting_time += current_time - processes[i].arrival_time;

        current_time += processes[i].burst_time;

        total_turnaround_time += current_time - processes[i].arrival_time;}

    printf("\n\nAverage Turnaround Time: %.2f", (float)total_turnaround_time / n);

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

    return 0;}
```

SLIP-20

Q.1 Write a program to create a child process using fork().The parent should goto sleep state and child process should begin its execution. In the child process, use execl() to execute the “ls” command.

ANSWER-1

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main() {
    pid_t child_pid = fork();
    if (child_pid < 0) {
        perror("Fork failed");
        exit(1);}
    if (child_pid == 0) {
        execl("/bin/ls", "ls", NULL);
        perror("Execl failed");
        exit(1);
    } else {
        printf("Parent process is going to sleep for 3 seconds...\n");
        sleep(3);
        wait(NULL);
        printf("Parent process has woken up.\n");}
    return 0;}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n=3 as the number of memory frames. Reference String : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 i. Implement LRU

ANSWER-2

```
#include <stdio.h>
#include <stdlib.h>
int findLRU(int frames[], int n, int page, int current) {
    int index = -1;
    int farthest = current;
    for (int i = 0; i < n; i++) {
        int j;
        for (j = current - 1; j >= 0; j--) {
            if (frames[i] == page) {
                break;}
            if (frames[i] == -1 || frames[i] == frames[j]) {
                if (j < farthest) {
                    farthest = j;
                    index = i;}
                break;}}}
    return index;}
int main() {
    int n = 3;
    int referenceString[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
    int referenceStringLength = sizeof(referenceString) / sizeof(referenceString[0]);
    int frames[n];
    int pageFaults = 0;
    for (int i = 0; i < n; i++) {

        frames[i] = -1;}

    printf("Page Scheduling using LRU:\n");

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

        int page = referenceString[i];

        int found = 0;

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

            if (frames[j] == page) {

                found = 1;

                break;}}

        if (!found) {

            int replaceIndex = findLRU(frames, n, page, i);

            frames[replaceIndex] = page;

            pageFaults++;

            printf("Page %d -> Page Fault\n", page);}}

    printf("Total Page Faults: %d\n", pageFaults);

    return 0;}
```