OS Lab Programs

Fibonacci Series:

```bash
#!/bin/bash

# Function to print Fibonacci series
fibonacci() {
    n=$1
    a=0
    b=1

    echo "Fibonacci series up to $n terms:"

    for (( i=0; i<n; i++ ))
    do
        echo -n "$a "
        fn=$((a + b))
        a=$b
        b=$fn
    done
    echo
}

# Read the number of terms from the user
read -p "Enter the number of terms: " num

# Call the Fibonacci function
fibonacci $num
```

Program to reverse a string

```bash
#!/bin/bash

# Read a string from the user
read -p "Enter a string: " str

# Get the length of the string
len=${#str}

# Initialize an empty string for the reversed string
reversed=""

# Loop through the string in reverse order
for (( i=$len-1; i>=0; i-- ))
do
```

```
    reversed="$reversed${str:$i:1}"
done

# Print the reversed string
echo "Reversed string: $reversed"
```

Factorial of a number

```
#!/bin/bash

# Read a number from the user
read -p "Enter a number: " num

# Initialize factorial to 1
factorial=1

# Loop to calculate factorial
for (( i=1; i<=num; i++ ))
do
    factorial=$((factorial * i))
done

# Print the factorial
echo "Factorial of $num is $factorial"
```

Find and display directories where the user has execute permissions

```
#!/bin/bash

# Find and display directories where the user has execute permissions
echo "Directories with execute permissions:"
find . -type d -perm -u=x
```

Find and remove duplicate files in the current directory

```
#!/bin/bash

# Find and remove duplicate files in the current directory
echo "Removing duplicate files..."

declare -A file_hashes

for file in *; do
    if [ -f "$file" ]; then
        hash=$(md5sum "$file" | awk '{ print $1 }')
        if [[ -n "${file_hashes[$hash]}" ]]; then
```

```
            echo "Removing duplicate file: $file"
            rm "$file"
        else
            file_hashes[$hash]=$file
        fi
    fi
done

echo "Duplicate removal complete."
```

FCFS

```
#include <stdio.h>

struct Process {
    int id;
    int arrivalTime;
    int burstTime;
    int waitingTime;
    int turnaroundTime;
};

void findWaitingTime(struct Process proc[], int n) {
    proc[0].waitingTime = 0; // First process has no waiting time

    for (int i = 1; i < n; i++) {
        proc[i].waitingTime = proc[i-1].waitingTime + proc[i-1].burstTime;
    }
}

void findTurnaroundTime(struct Process proc[], int n) {
    for (int i = 0; i < n; i++) {
        proc[i].turnaroundTime = proc[i].waitingTime + proc[i].burstTime;
    }
}

void findAvgTime(struct Process proc[], int n) {
    findWaitingTime(proc, n);
    findTurnaroundTime(proc, n);

    printf("Processes  Arrival Time  Burst Time  Waiting Time  Turnaround 
Time\n");
    for (int i = 0; i < n; i++) {
        printf("   %d\t\t%d\t\t%d\t\t%d\t\t%d\n", proc[i].id, 
proc[i].arrivalTime, proc[i].burstTime, proc[i].waitingTime, 
proc[i].turnaroundTime);
    }
}
```

```c

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

    struct Process proc[n];
    for (int i = 0; i < n; i++) {
        proc[i].id = i+1;
        printf("Enter arrival time and burst time for process %d: ", i+1);
        scanf("%d %d", &proc[i].arrivalTime, &proc[i].burstTime);
    }

    findAvgTime(proc, n);

    return 0;
}
```

Round Robin

```c
#include <stdio.h>

struct Process {
    int id;
    int arrivalTime;
    int burstTime;
    int remainingTime;
    int waitingTime;
    int turnaroundTime;
};

void findWaitingTime(struct Process proc[], int n, int quantum) {
    int time = 0;
    int done;

    do {
        done = 1;
        for (int i = 0; i < n; i++) {
            if (proc[i].remainingTime > 0) {
                done = 0;
                if (proc[i].remainingTime > quantum) {
                    time += quantum;
                    proc[i].remainingTime -= quantum;
                } else {
                    time += proc[i].remainingTime;
                    proc[i].waitingTime = time - proc[i].burstTime;
                    proc[i].remainingTime = 0;
                }
            }
        }
    } while (!done);
}

void findTurnaroundTime(struct Process proc[], int n) {
    for (int i = 0; i < n; i++) {
        proc[i].turnaroundTime = proc[i].burstTime + proc[i].waitingTime;
    }
}

void findAvgTime(struct Process proc[], int n, int quantum) {
    findWaitingTime(proc, n, quantum);
    findTurnaroundTime(proc, n);

    printf("Processes  Arrival Time  Burst Time  Waiting Time  Turnaround
Time\n");
    for (int i = 0; i < n; i++) {
        printf("   %d\t\t%d\t\t%d\t\t%d\t\t%d\n", proc[i].id,
proc[i].arrivalTime, proc[i].burstTime, proc[i].waitingTime,
proc[i].turnaroundTime);
    }
}

int main() {
    int n, quantum;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process proc[n];
    for (int i = 0; i < n; i++) {
        proc[i].id = i + 1;
        printf("Enter arrival time and burst time for process %d: ", i + 1);
        scanf("%d %d", &proc[i].arrivalTime, &proc[i].burstTime);
        proc[i].remainingTime = proc[i].burstTime;
    }

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

    findAvgTime(proc, n, quantum);

    return 0;
}
```

SJF

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

```c
struct Process {
    int id;
    int arrivalTime;
    int burstTime;
    int waitingTime;
    int turnaroundTime;
};

void findWaitingTime(struct Process proc[], int n) {
    int completed = 0, time = 0, minBurst = 9999, shortest = 0;
    int finishTime;
    int check = 0;

    while (completed != n) {
        for (int j = 0; j < n; j++) {
            if ((proc[j].arrivalTime <= time) && (proc[j].burstTime <
minBurst) && proc[j].burstTime > 0) {
                minBurst = proc[j].burstTime;
                shortest = j;
                check = 1;
            }
        }

        if (check == 0) {
            time++;
            continue;
        }

        proc[shortest].burstTime--;
        minBurst = proc[shortest].burstTime;
        if (minBurst == 0) {
            minBurst = 9999;
        }

        if (proc[shortest].burstTime == 0) {
            completed++;
            check = 0;
            finishTime = time + 1;
            proc[shortest].waitingTime = finishTime -
proc[shortest].arrivalTime - proc[shortest].burstTime;
            if (proc[shortest].waitingTime < 0) {
                proc[shortest].waitingTime = 0;
            }
        }
        time++;
    }
}
```

```c
void findTurnaroundTime(struct Process proc[], int n) {
    for (int i = 0; i < n; i++) {
        proc[i].turnaroundTime = proc[i].burstTime + proc[i].waitingTime;
    }
}

void findAvgTime(struct Process proc[], int n) {
    findWaitingTime(proc, n);
    findTurnaroundTime(proc, n);

    printf("Processes  Arrival Time  Burst Time  Waiting Time  Turnaround 
Time\n");
    for (int i = 0; i < n; i++) {
        printf("   %d\t\t%d\t\t%d\t\t%d\t\t%d\n", proc[i].id, 
proc[i].arrivalTime, proc[i].burstTime, proc[i].waitingTime, 
proc[i].turnaroundTime);
    }
}

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

    struct Process proc[n];
    for (int i = 0; i < n; i++) {
        proc[i].id = i + 1;
        printf("Enter arrival time and burst time for process %d: ", i + 1);
        scanf("%d %d", &proc[i].arrivalTime, &proc[i].burstTime);
    }

    findAvgTime(proc, n);

    return 0;
}
```

Priority Scheduling

```c
#include <stdio.h>

struct Process {
    int id;
    int arrivalTime;
    int burstTime;
    int priority;
    int waitingTime;
    int turnaroundTime;
```

```c
};

void findWaitingTime(struct Process proc[], int n) {
    proc[0].waitingTime = 0; // First process has no waiting time

    for (int i = 1; i < n; i++) {
        proc[i].waitingTime = proc[i-1].waitingTime + proc[i-1].burstTime;
    }
}

void findTurnaroundTime(struct Process proc[], int n) {
    for (int i = 0; i < n; i++) {
        proc[i].turnaroundTime = proc[i].waitingTime + proc[i].burstTime;
    }
}

void findAvgTime(struct Process proc[], int n) {
    findWaitingTime(proc, n);
    findTurnaroundTime(proc, n);

    printf("Processes  Arrival Time  Burst Time  Priority  Waiting
Time  Turnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("   %d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", proc[i].id,
proc[i].arrivalTime, proc[i].burstTime, proc[i].priority, proc[i].waitingTime,
proc[i].turnaroundTime);
    }
}

void priorityScheduling(struct Process proc[], int n) {
    struct Process temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (proc[i].priority > proc[j].priority) {
                temp = proc[i];
                proc[i] = proc[j];
                proc[j] = temp;
            }
        }
    }
}

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

    struct Process proc[n];
    for (int i = 0; i < n; i++) {
        proc[i].id = i + 1;
        printf("Enter arrival time, burst time and priority for process %d: ",
i + 1);
        scanf("%d %d %d", &proc[i].arrivalTime, &proc[i].burstTime,
&proc[i].priority);
    }

    priorityScheduling(proc, n);
    findAvgTime(proc, n);

    return 0;
}
```