21BDS0340

Abhinav Dinesh Srivatsa

Operating Systems Lab

Assignement – III

**Practice:**

* Finding area and circumference of a circle

Program:

echo Enter radius:

read n

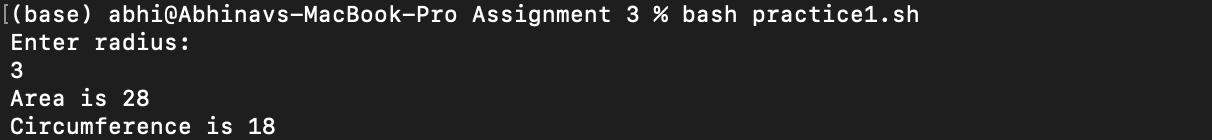
area=`expr $n \\* $n \\* 22 / 7`

circ=`expr 2 \\* $n \\* 22 / 7`

echo Area is $area

echo Circumference is $circ

Output:



* Finding the sum of the first 5 natural numbers

Program:

sum=0

echo First 5 natural numbers

for i in 1 2 3 4 5

do

sum=`expr $sum + $i`

echo $i

done

echo The sum is $sum

Output:

A black screen with white text

Description automatically generated with low confidence

**Assignment:**

**Question 1**

* First come first served (FCFS)

Program:

arrival=()

burst=()

echo Enter arrival times:

read input

read -a arrival <<< "$input"

echo Enter burst times:

read input

read -a burst <<< "$input"

if [ ${#arrival[@]} -eq ${#burst[@]} ]

then

arrival\_len=${#arrival[@]}

# selection sort by arrival time

for((i=0; i<arrival\_len-1; i++));

do

min=$i

for ((j=i+1; j<arrival\_len; j++))

do

if ((arrival[j] < arrival[min]))

then min=$j

fi

done

# swapping arrival time

temp=${arrival[i]}

arrival[i]=${arrival[min]}

arrival[min]=$temp

# swapping burst times

temp=${burst[i]}

burst[i]=${burst[min]}

burst[min]=$temp

done

time=${arrival[0]}

# display no process if time doesnt start at 0

if ((time != 0))

then echo 0 - $time: No process in ready queue

fi

# display results

for i in ${!arrival[@]}

do

echo $time - $((time + burst[i])): Process with arrival time: ${arrival[i]} and burst time: ${burst[i]}

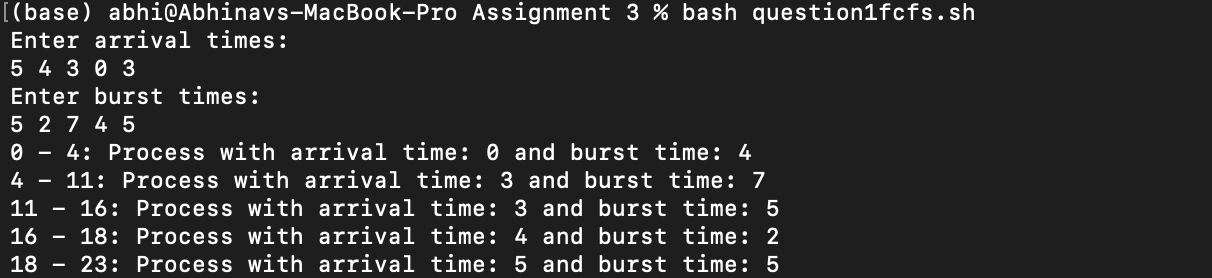
time=$((time + burst[i]))

done

else echo Number of values must be equal!

fi

Output:



* Shortest job first (SJF)

Program:

arrival=()

burst=()

ready=()

completed=()

echo Enter arrival times:

read input

read -a arrival <<< "$input"

echo Enter burst times:

read input

read -a burst <<< "$input"

if [ ${#arrival[@]} -eq ${#burst[@]} ]

then

arrival\_len=${#arrival[@]}

# selection sort by arrival time then burst time

for((i=0; i<arrival\_len-1; i++));

do

min=$i

for ((j=i+1; j<arrival\_len; j++))

do

if ((arrival[j] < arrival[min]))

then min=$j

fi

if ((arrival[j] == arrival[min]))

then if ((burst[j] < burst[min]))

then min=$j

fi

fi

done

# swapping arrival time

temp=${arrival[i]}

arrival[i]=${arrival[min]}

arrival[min]=$temp

# swapping burst times

temp=${burst[i]}

burst[i]=${burst[min]}

burst[min]=$temp

done

time=${arrival[0]}

# looping times for number of processes

for ((n=0; n<arrival\_len; n++))

do

# building ready queue

for ((i=0; i<arrival\_len; i++))

do

if ((arrival[i] <= time))

then

# checking if process is marked completed or already

process\_complete=0

completed\_len=${#completed[@]}

for ((j=0; j<completed\_len; j++))

do

if ((i == completed[j]))

then process\_complete=1

fi

done

if ((process\_complete == 0))

then ready+=($i)

fi

fi

done

# finding min burst time in ready queue

min\_burst=0

ready\_len=${#ready[@]}

for ((i=0; i<ready\_len; i++))

do

if ((burst[ready[i]] < burst[ready[min\_burst]]))

then min\_burst=$i

fi

done

# displaying process info

echo $time - $((time + burst[ready[min\_burst]])): Process with arrival time: ${arrival[ready[min\_burst]]} and burst time: ${burst[ready[min\_burst]]}

time=$((time + burst[ready[min\_burst]]))

# marking process as completed

completed+=($((ready[min\_burst])))

# clearing ready queue

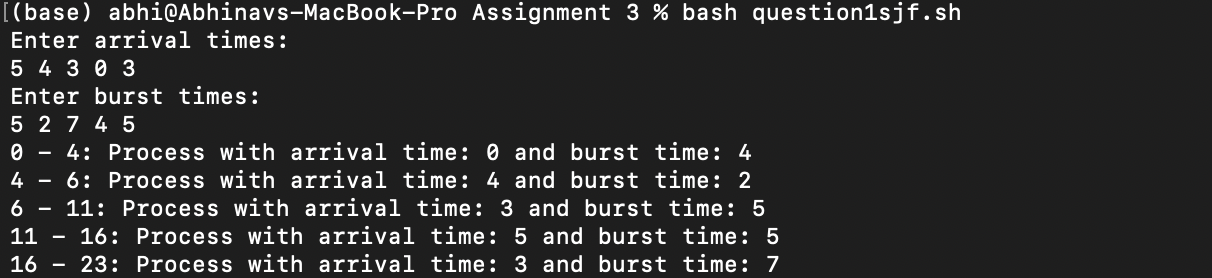
ready=()

done

else echo Number of values must be equal!

fi

Output:



* Priority

Program:

arrival=()

burst=()

priority=()

ready=()

completed=()

echo Enter arrival times:

read input

read -a arrival <<< "$input"

echo Enter burst times:

read input

read -a burst <<< "$input"

echo Enter priority times:

read input

read -a priority <<< "$input"

if [ ${#arrival[@]} -eq ${#burst[@]} ]

then

arrival\_len=${#arrival[@]}

# selection sort by priority and arrival time

for((i=0; i<arrival\_len-1; i++));

do

min=$i

for ((j=i+1; j<arrival\_len; j++))

do

if ((priority[j] < priority[min]))

then min=$j

fi

if ((priority[j] == priority[min]))

then if ((arrival[j] < arrival[min]))

then min=$j

fi

fi

done

# swapping arrival time

temp=${arrival[i]}

arrival[i]=${arrival[min]}

arrival[min]=$temp

# swapping burst times

temp=${burst[i]}

burst[i]=${burst[min]}

burst[min]=$temp

# swapping priorities

temp=${priority[i]}

priority[i]=${priority[min]}

priority[min]=$temp

done

time=${arrival[0]}

# looping times for number of processes

for ((n=0; n<arrival\_len; n++))

do

# building ready queue

for ((i=0; i<arrival\_len; i++))

do

if ((arrival[i] <= time))

then

# checking if process is marked completed or already

process\_complete=0

completed\_len=${#completed[@]}

for ((j=0; j<completed\_len; j++))

do

if ((i == completed[j]))

then process\_complete=1

fi

done

if ((process\_complete == 0))

then ready+=($i)

fi

fi

done

# finding min burst time in ready queue

min\_burst=0

ready\_len=${#ready[@]}

for ((i=0; i<ready\_len; i++))

do

if ((priority[ready[i]] > priority[ready[min\_burst]]))

then min\_burst=$i

fi

done

# displaying process info

echo $time - $((time + burst[ready[min\_burst]])): Process with arrival time: ${arrival[ready[min\_burst]]} and burst time: ${burst[ready[min\_burst]]} and priority: ${priority[ready[min\_burst]]}

time=$((time + burst[ready[min\_burst]]))

# marking process as completed

completed+=($((ready[min\_burst])))

# clearing ready queue

ready=()

done

else echo Number of values must be equal!

fi

Output:

A screen shot of a computer

Description automatically generated with medium confidence

* Round Robin (non-preemptive)

Program:

arrival=()

burst=()

echo Enter arrival times:

read input

read -a arrival <<< "$input"

echo Enter burst times:

read input

read -a burst <<< "$input"

if [ ${#arrival[@]} -eq ${#burst[@]} ]

then

arrival\_len=${#arrival[@]}

# selection sort by arrival time

for((i=0; i<arrival\_len-1; i++));

do

min=$i

for ((j=i+1; j<arrival\_len; j++))

do

if ((arrival[j] < arrival[min]))

then min=$j

fi

done

# swapping arrival time

temp=${arrival[i]}

arrival[i]=${arrival[min]}

arrival[min]=$temp

# swapping burst times

temp=${burst[i]}

burst[i]=${burst[min]}

burst[min]=$temp

done

time=${arrival[0]}

# display no process if time doesnt start at 0

if ((time != 0))

then echo 0 - $time: No process in ready queue

fi

# display results

for i in ${!arrival[@]}

do

echo $time - $((time + burst[i])): Process with arrival time: ${arrival[i]} and burst time: ${burst[i]}

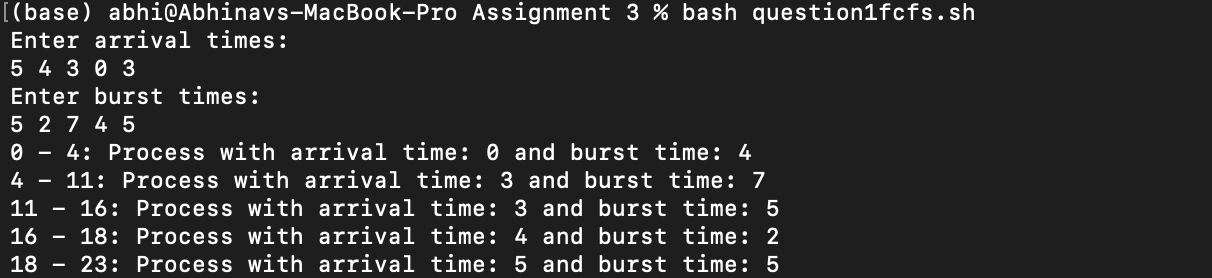
time=$((time + burst[i]))

done

else echo Number of values must be equal!

fi

Output:



**Question 2**

Program

echo Enter number of processes:

read processes

echo Enter types of resources:

read resource\_types

free\_resources=()

assigned=()

required=()

# reading free resources

echo Enter free resources:

read input

read -a free\_resources <<< $input

# reading assigned resources

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

do

echo "Enter space-separated assigned resources for process $((i + 1)):"

read -ra row\_input

assigned+=(${row\_input[@]})

done

# reading required resources

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

do

echo "Enter space-separated required resources for process $((i + 1)):"

read -ra row\_input

required+=(${row\_input[@]})

done

completed=()

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

do

# checking if enough resources are available for process

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

do

# checking if process already completed

is\_completed=0

for j in ${completed[@]}

do

if ((j == i))

then is\_completed=1

fi

done

if ((is\_completed == 0))

then

completable=1

for ((x=0; x<resource\_types; x++))

do

r=`expr $resource\_types \\* $i + $x`

if ((assigned[r] + free\_resources[x] < required[r]))

then completable=0

fi

done

if ((completable == 1))

then

for ((x=0; x<resource\_types; x++))

do

free\_resources[x]=$((free\_resources[x] + assigned[r]))

done

completed+=($i)

fi

fi

done

done

completed\_len=${#completed[@]}

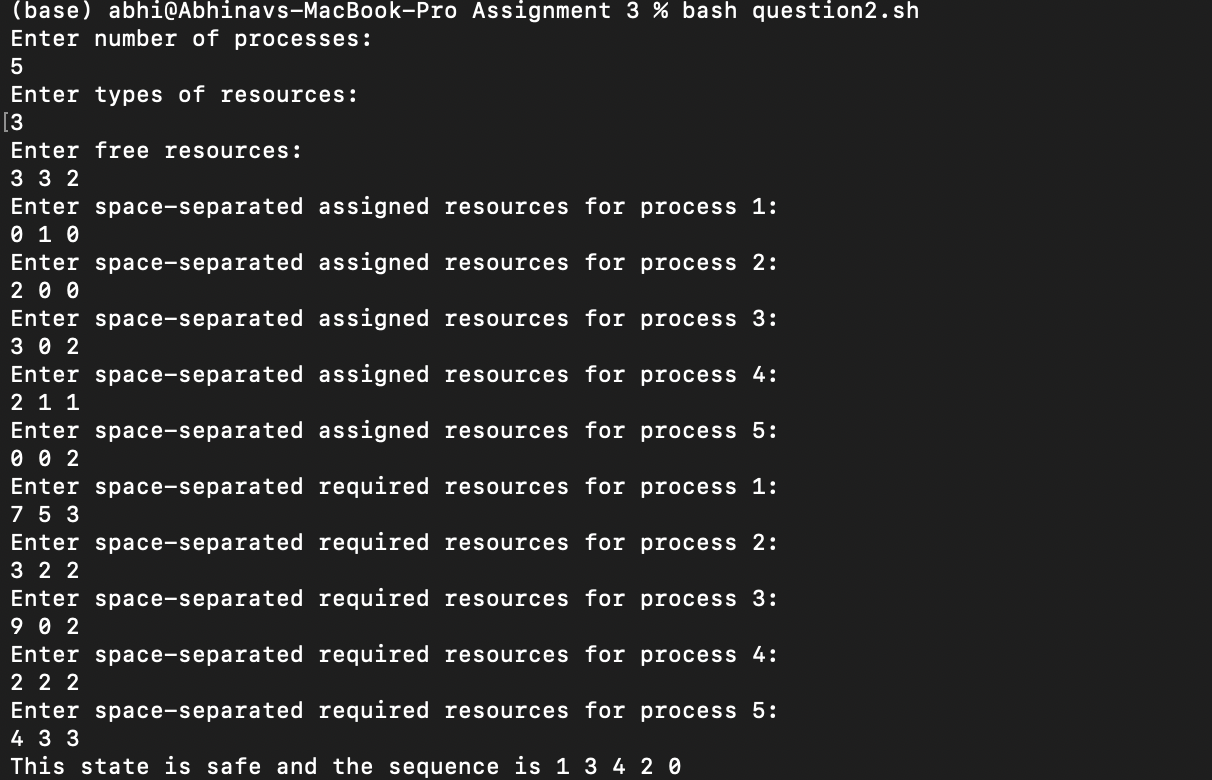
if ((completed\_len == processes))

then echo This state is safe and the sequence is ${completed[@]}

else echo This state is unsafe

fi

Output:



**Question 3**

Program:

semaphore=1

wait(){

while ((semaphore <= 0))

do semaphore=$semaphore # do nothing

done

semaphore=$((semaphore - 1))

sleep 5 # simulating process working

signal

}

signal(){

semaphore=$((semaphore + 1))

}

# executing process 1

echo Started process 1...

wait

echo Process 1 completed.

echo ""

# wait and then execute process 2

echo Started process 2...

wait

echo Process 2 completed.

Output:

