Youser Alalusi
HW#4
Silver Bullet
11/13/2021

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


// keeps track of the number of processes
int numOfProcesses;


// To create a process, holds all of the information
struct process {

// id to identify process
int id;

// time needed to run to completion
int timeNeeded;

// priority of process (for HPF only)
int priority;

//marker to show if process has been used already (HPF only)
bool usedPriority;

// amount of slices needed to complete process (for RR only)
int slices;

// total time in ready queue
int waitTime;
int turnAround;
int totalDuration;
};


// array to hold raw input by user, not sorted by scheduler
struct process rawInput[];
```

Youser Alalusi
HW#4
Silver Bullet
11/13/2021

```
// First come first serve alg

void fcfs() {
// ready queue, will hold all of the processes in the scheduled order
struct process readyQueue[numOfProcesses];
double totalTime = 0;
double totalWait = 0;
double totalTurnaround = 0;
float throughput;


// Place processes in the ready queue by the order they are inputted
for (int i = 0; i < numOfProcesses; i++) {
   readyQueue[i] = rawInput[i];
}


printf("\nProcess list in FCFS order entered:\n");
for (int i = 0; i < numOfProcesses; i++) {
   printf("%d %d %d\n", readyQueue[i].id, readyQueue[i].timeNeeded, readyQueue[i].priority);
   readyQueue[i].waitTime += totalTime;
   readyQueue[i].turnAround += readyQueue[i].waitTime + readyQueue[i].timeNeeded;
   totalTime += readyQueue[i].timeNeeded;
}
printf("End of list.\n\n");

for (int i = 0; i < numOfProcesses; i++) {
   printf("fcfs wait of p%d = %d\n", readyQueue[i].id, readyQueue[i].waitTime);
   totalWait += readyQueue[i].waitTime;
}
printf("average wait time for %d procs = %0.1f\n", numOfProcesses, (double)
(totalWait/numOfProcesses));
for (int i = 0; i < numOfProcesses; i++) {
   printf("fcfs turn-around time for p%d = %d\n", readyQueue[i].id, readyQueue[i].turnAround);
   totalTurnaround += readyQueue[i].turnAround;
}
printf("average turn-around for %d procs = %0.1f\n", numOfProcesses, (double) (totalTurnaround /
numOfProcesses));
printf("fcfs throughput for %d procs = %f proc/ms\n", numOfProcesses, (double)
(numOfProcesses/totalTime));
printf("\nend FCFS schedule \n");
}
```

Youser Alalusi
HW#4
Silver Bullet
11/13/2021

```
// Highest priority sorting alg

void hpf() {

// ready queue, holds all of the processes in the scheduled order
struct process readyQueue[numOfProcesses];
int readyIndex = 0
double totalTime = 0;
double totalWait = 0;
double totalTurnaround = 0;
float throughput;
int currentPriority = 100;
int lastUsedPriority = -1;
int usedIndex;
struct process currentHighestProcess

// Sorting by priority
while (readyIndex != numOfProcesses) {

   for (int i = 0; i < numOfProcesses; i++) {
      if ((rawInput[i].priority < currentPriority) && (rawInput[i].priority > lastUsedPriority) &&
(rawInput[i].usedPriority != true)) {
         currentPriority = rawInput[i].priority;
         currentHighestProcess = rawInput[i];
         usedIndex = i;
      }
   }
   readyQueue[readyIndex] = currentHighestProcess;
   currentPriority = 100;
   lastUsedPriority = currentHighestProcess.priority;
   rawInput[usedIndex].usedPriority = true;
   readyIndex++;
}
printf("\nProcess list in HPF order:\n");
for (int i = 0; i < numOfProcesses; i++) {
   printf("%d %d %d\n", readyQueue[i].id, readyQueue[i].timeNeeded, readyQueue[i].priority);
   readyQueue[i].waitTime += totalTime;
   readyQueue[i].turnAround += readyQueue[i].waitTime + readyQueue[i].timeNeeded;
   totalTime += readyQueue[i].timeNeeded;
}
```

```
printf("End of list.\n\n");
for (int i = 0; i < numOfProcesses; i++) {
    printf("hpf wait of p%d = %d\n", readyQueue[i].id, readyQueue[i].waitTime);
    totalWait += readyQueue[i].waitTime;
}
printf("average wait time for %d procs = %0.1f\n", numOfProcesses, (double)
(totalWait/numOfProcesses));
for (int i = 0; i < numOfProcesses; i++) {
    printf("hpf turn-around time for p%d = %d\n", readyQueue[i].id, readyQueue[i].turnAround);
    totalTurnaround += readyQueue[i].turnAround;
}
printf("average turn-around for %d procs = %0.1f\n", numOfProcesses, (double) (totalTurnaround /
numOfProcesses));
printf("hpf throughput for %d procs = %f proc/ms\n", numOfProcesses, (double)
(numOfProcesses/totalTime));
printf("\nend HPF schedule\n");
}


// Round robin sorting alg
void roundRobin() {
 // ready queue, holds all of the processes in the scheduled order
struct process readyQueue[numOfProcesses];
double totalTime = 0;
double totalTimeLeft = 0;
double completionTime = 0;
double avgTime;
float throughput;
double totalTurnaround;
bool allComplete = false;
for (int i = 0; i < numOfProcesses; i++) {
    readyQueue[i] = rawInput[i];
}
```

Youser Alalusi
HW#4
Silver Bullet
11/13/2021

```
// Place processes in the ready queue by the order they are inputted and print the process list.
printf("\nProcess list for RR in order entered:\n");
for (int i = 0; i < numOfProcesses; i++) {
  printf("%d %d %d\n", readyQueue[i].id, readyQueue[i].timeNeeded, readyQueue[i].priority);
}
printf("End of list.\n");
for(int quantum = 1; quantum <=5; ++quantum){
  for(int overhead = 0; overhead <=quantum; ++overhead){
    for (int i = 0; i < numOfProcesses; i++) {
      readyQueue[i] = rawInput[i];
    }
    allComplete = false;
    totalTime = 0;
    completionTime = 0;
    printf("\npreemptive RR schedule, quantum = %d overhead = %d\n", quantum, overhead);
    while (!allComplete){
      for (int i = 0; i < numOfProcesses; i++) {
        if (readyQueue[i].timeNeeded == 0) {
          continue;
        }
        else {
          if(i == 0 && totalTime == 0){
            if(readyQueue[i].timeNeeded < quantum){
              totalTime += readyQueue[i].timeNeeded;
              readyQueue[i].slices++;
              readyQueue[i].timeNeeded = 0;
            }
            else{
              totalTime += quantum;
              readyQueue[i].slices++;
              readyQueue[i].timeNeeded -= quantum;
            }
          }
          else if (readyQueue[i].timeNeeded - quantum == 0 || readyQueue[i].timeNeeded < quantum)
{
            totalTime += readyQueue[i].timeNeeded + overhead;
            readyQueue[i].slices++;
            readyQueue[i].timeNeeded = 0;
          }
          else {
            totalTime += quantum + overhead;
            readyQueue[i].slices++;
            readyQueue[i].timeNeeded -= quantum;
```

```
                    }
                    readyQueue[i].waitTime = totalTime;
                }
            }
            totalTimeLeft = 0;
            for (int i = 0; i < numOfProcesses; i++) {
                totalTimeLeft += readyQueue[i].timeNeeded;
            }
            if (totalTimeLeft == 0) {
                allComplete = true;
            } else {
                allComplete = false;
                totalTimeLeft = 0;
            }
        }
        totalTurnaround = 0;
        int k, l;
        for (k = 0; k < numOfProcesses-1; k++){
            for (l = 0; l < numOfProcesses-k-1; l++){
                if (readyQueue[l].waitTime > readyQueue[l+1].waitTime){
                    struct process temp;
                    temp = readyQueue[l];
                    readyQueue[l] = readyQueue[l+1];
                    readyQueue[l+1] = temp;
                }
            }
        }
        for (int i = 0; i < numOfProcesses; i++) {
            readyQueue[i].turnAround = readyQueue[i].waitTime;
            printf("RR TA time for finished p%d = %d, needed: %d ms, and: %d time slices.\n", 
readyQueue[i].id, readyQueue[i].turnAround, readyQueue[i].totalDuration, readyQueue[i].slices);
            totalTurnaround += readyQueue[i].turnAround;
            if (completionTime < readyQueue[i].turnAround) {
                completionTime = readyQueue[i].turnAround;
            }
        }
        printf("RR Throughput, %d p, with q: %d, o: %d, is: %0.4f p/ms, or %0.4f p/us\n", numOfProcesses, 
quantum, overhead, (double) (numOfProcesses/completionTime), (float) 
(1000*(numOfProcesses/completionTime)));
        printf("Average RR TA, %d p, with q: %d, o: %d, is: %0.4f\n", numOfProcesses, quantum, overhead, 
(float) (totalTurnaround / numOfProcesses));
    }
}
```

```
printf("\n end preemptive RR schedule \n");
}

int main(int argc, char* argv[]) {
int x, y, z;
if (argc != 1) {
  fprintf(stderr, "Error: Please enter [out_file name]");
  exit(-1);
} else {
  printf("Enter triples: process id, time in ms, and priority. Enter 'end' when done\n");
  printf("For example:\n");
  printf("1 12 0\n");
  printf("3  9 1\n");
  printf("2 99 9\n");
  printf("process 1 needs 12 ms and has priority 0 (highest)\n");
  printf("process 3 needs  9 ms and has priority 1\n");
  printf("process 2 needs 99 ms and has priority 9\n");
}
  while (scanf("%d %d %d", &x, &y, &z) == 3) {
    numOfProcesses++;
    rawInput[numOfProcesses-1].id = x;
    rawInput[numOfProcesses-1].timeNeeded = y;
    rawInput[numOfProcesses-1].priority = z;
    rawInput[numOfProcesses-1].usedPriority = false;
    rawInput[numOfProcesses-1].slices = 0;
    rawInput[numOfProcesses-1].waitTime = 0;
    rawInput[numOfProcesses-1].turnAround = 0;
    rawInput[numOfProcesses-1].totalDuration = y;

    /*printf("process id = %d\n", rawInput[numOfProcesses-1].id);
    printf("time needed = %d\n", rawInput[numOfProcesses-1].timeNeeded);
    printf("priority = %d\n", rawInput[numOfProcesses-1].priority);*/
  }

  fcfs();
  hpf();
  roundRobin();
  //return 0;
}
```