Operating Systems Homework #3

김현준 (2012003954), 한양대학교 2015-05-14

Process Scheduling Simulator

References:

1. Full HW description:

   https://github.com/yoloseem/os-homeworks/blob/master/hw3/README.md

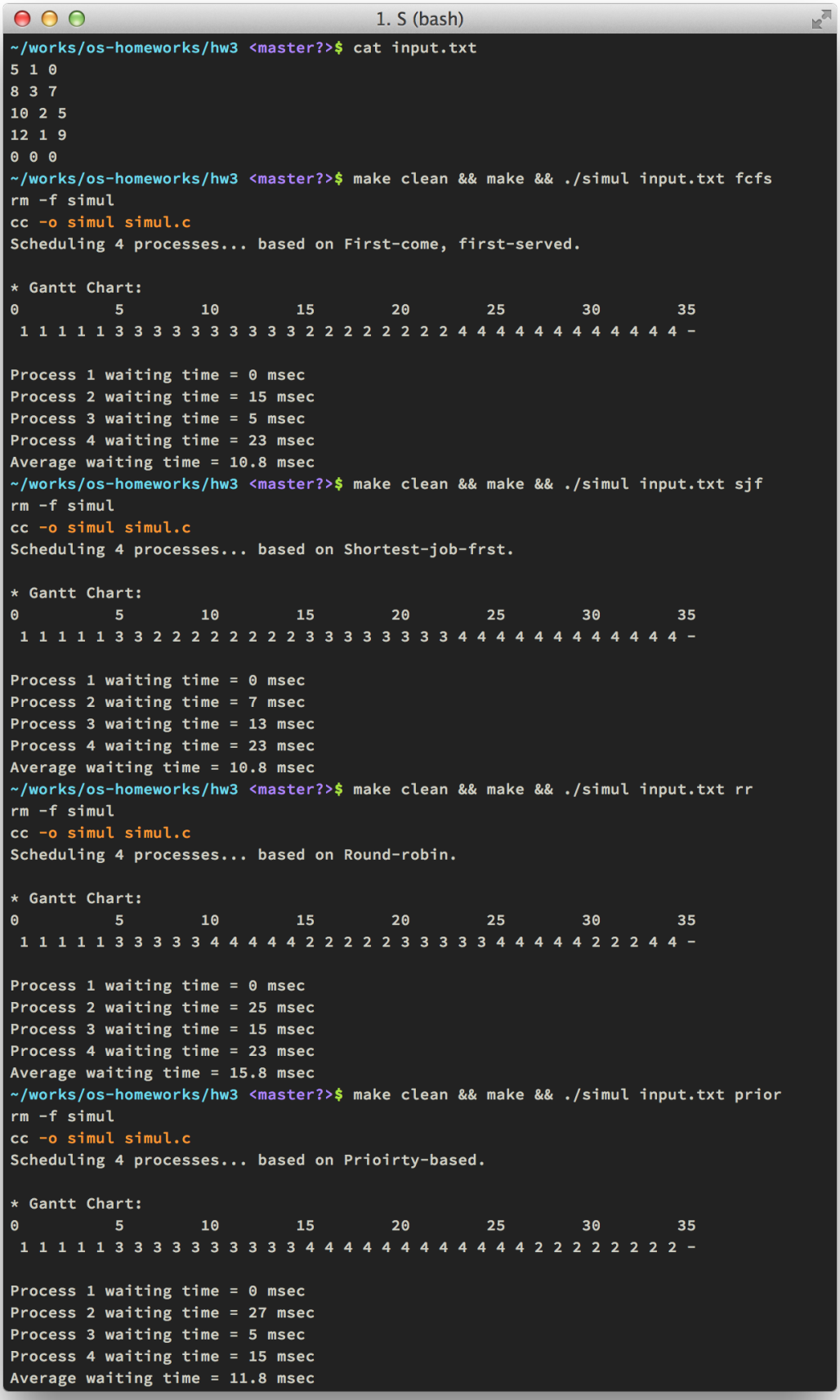
2. Raw source codes:

   https://github.com/yoloseem/os-homeworks/tree/master/hw3

3. Commit history:

   https://github.com/yoloseem/os-homeworks/commits/master

Screenshot:


1. S (bash)
~/works/os-homeworks/hw3 <master?>$ cat input.txt
5 1 0
8 3 7
10 2 5
12 1 9
0 0 0
~/works/os-homeworks/hw3 <master?>$ make clean && make && ./simul input.txt fcfs
rm -f simul
cc -o simul simul.c
Scheduling 4 processes... based on First-come, first-served.
* Gantt Chart:
0 5 10 15 20 25 30 35
1 1 1 1 1 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 4 4 4 4 4 4 4 4 4 4 4 4 4 -
Process 1 waiting time = 0 msec
Process 2 waiting time = 15 msec
Process 3 waiting time = 5 msec
Process 4 waiting time = 23 msec
Average waiting time = 10.8 msec
~/works/os-homeworks/hw3 <master?>$ make clean && make && ./simul input.txt sjf
rm -f simul
cc -o simul simul.c
Scheduling 4 processes... based on Shortest-job-frst.
* Gantt Chart:
0 5 10 15 20 25 30 35
1 1 1 1 1 3 3 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 -
Process 1 waiting time = 0 msec
Process 2 waiting time = 7 msec
Process 3 waiting time = 13 msec
Process 4 waiting time = 23 msec
Average waiting time = 10.8 msec
~/works/os-homeworks/hw3 <master?>$ make clean && make && ./simul input.txt rr
rm -f simul
cc -o simul simul.c
Scheduling 4 processes... based on Round-robin.
* Gantt Chart:
0 5 10 15 20 25 30 35
1 1 1 1 1 3 3 3 3 3 4 4 4 4 4 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4 4 2 2 2 4 4 -
Process 1 waiting time = 0 msec
Process 2 waiting time = 25 msec
Process 3 waiting time = 15 msec
Process 4 waiting time = 23 msec
Average waiting time = 15.8 msec
~/works/os-homeworks/hw3 <master?>$ make clean && make && ./simul input.txt prior
rm -f simul
cc -o simul simul.c
Scheduling 4 processes... based on Prioirty-based.
* Gantt Chart:
0 5 10 15 20 25 30 35
1 1 1 1 1 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 2 2 2 2 2 2 2 2 -
Process 1 waiting time = 0 msec
Process 2 waiting time = 27 msec
Process 3 waiting time = 5 msec
Process 4 waiting time = 15 msec
Average waiting time = 11.8 msec

Source codes:

Makefile

```
# Makefile
#
all: simul

clean:
	${RM} simul

simul:
	${CC} -o simul simul.c
```

simul.c (Main source code)

```
/* simul.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define ERROREXIT() {printf("ERROR!\n"); exit(0);}

#define MAX_PROCESSES 10
#define MAX_TIMELAPSE 100
#define RR_QUANTUM 5

typedef struct Process {
    // (Integer) Bursting time in milliseconds
    int burstTime;
    // (Integer) Priority (lower value = higher prioirty)
    int priority;
    // (Integer) Time at the process appeared/created
    int startAt;
    // (Integer) Waiting time for the process
    int waitTime;
} Process;
Process procs[MAX_PROCESSES];

// (Integer) Number of process to be scheduled
int n;
// (Integer) Number indicating scheduling algorithm
int policy;
// Enum values for policy
const short FCFS=0, SJF=1, PRIOR=2, RR=3;
char *verbosePolicy[] = {"First-come, first-served",
                         "Shortest-job-frst",
                         "Prioirty-based",
                         "Round-robin"};

short gantt[MAX_TIMELAPSE];

```

```
37  int main (int argc, char** argv) {
38      if (argc == 3) {
39          /* Input from text file
40           *      $ ./exename [filename] [policy]
41           *      ([policy] can be one of 'fcfs', 'sjf', 'prior', or 'rr')
42           *
43           * Text file must be in format of:
44           *      Each line contains: "[burst time] [priority] [start time]"
45           *      "0 0 0" indicates the end of the input */
46          FILE *inputFp = fopen(argv[1], "r");
47          while( inputFp ) {
48              fscanf(inputFp, "%d%d%d",
49                     &procs[n].burstTime, &procs[n].priority, &procs[n].startAt);
50              if (!(procs[n].burstTime | procs[n].priority | procs[n].startAt))
51                  break;
52              procs[n].waitTime = 0;
53              n++;
54          }
55          if (!n) ERROREXIT();
56      }
57      else {
58          printf("Execute the program in format of:\n");
59          printf("    $ %s [filename] [policy]\n", argv[0]);
60          printf("    ([policy] can be one of ");
61          printf("'fcfs', 'sjf', 'prior', or 'rr')\n");
62      }
63
64      printf("Scheduling %d processes... ", n);
65      if (!strcmp(argv[2], "fcfs")) policy = FCFS;
66      else if (!strcmp(argv[2], "sjf")) policy = SJF;
67      else if (!strcmp(argv[2], "prior")) policy = PRIOR;
68      else if (!strcmp(argv[2], "rr")) policy = RR;
69      else {
70          printf("policy must be one of 'fcfs', 'sjf', 'prior', or 'rr'\n");
71          ERROREXIT();
72      }
73      printf("based on %s.\n", verbosePolicy[policy]);
74
75      int i, timelapsed = 0;
76      int quantum = RR_QUANTUM;
77      int pick = -1;
78      int futureProc = 0;
79
80      do { // Repeat until there's no process that has remaining burst
81          futureProc = 0;
82
83          /* Picking process to run in next single millisecond
84           * based on given scheduling policy */
85          if (policy == FCFS) {
86              pick = -1;
87              int firstStartAt = 0x7fffffff;
```

```
88              /* FCFS's picking criteria: first come (startAt) */
89              for (i=0; i<n; i++) {
90                  if (procs[i].burstTime <= 0) continue;
91                  if (procs[i].startAt > timelapsed) {
92                      futureProc = 1;
93                      continue;
94                  }
95
96                  if (firstStartAt > procs[i].startAt) {
97                      firstStartAt = procs[i].startAt;
98                      pick = i;
99                  }
100             }
101         }
102         else if (policy == RR) {
103             /* RR's picking: switch to next only when current time quantum has
104              * been ended */
105             if (pick == -1) pick = 0;
106             if (quantum == 0) {
107                 quantum = RR_QUANTUM;
108                 pick++;
109             }
110             for (i=0; i<n; i++) {
111                 if (procs[(pick + i) % n].burstTime > 0) {
112                     if (procs[(pick + i) % n].startAt > timelapsed) {
113                         futureProc = 1;
114                         continue;
115                     }
116                     pick = (pick + i) % n;
117                     quantum--;
118                     break;
119                 }
120             }
121             if (i == n) {
122                 pick = -1;
123                 quantum = 0;
124             }
125         }
126         else if (policy == PRIOR) {
127             /* PRIOR's picking: Highest priority first (preemptive,
128              * lower value is higher priority */
129             int highprior = 0x7fffffff;
130             if (pick != -1 && procs[pick].burstTime == 0) pick = -1;
131             if (pick == -1) {
132                 for (i=0; i<n; i++) {
133                     if (procs[i].burstTime > 0) {
134                         if (procs[i].startAt > timelapsed) {
135                             futureProc = 1;
136                             continue;
137                         }
138                         if (highprior > procs[i].priority) {
```

```
                            highprior = procs[i].priority;
                            pick = i;
                        }
                    }
                }
            }
        }
        else if (policy == SJF) {
            pick = -1;
            int shortestBurst = 0x7fffffff;
            /* SJF's picking criteria: shortest remaining burst (burstTime) */
            for (i=0; i<n; i++) {
                if (procs[i].burstTime <= 0) continue;
                if (procs[i].startAt > timelapsed) {
                    futureProc = 1;
                    continue;
                }

                if (shortestBurst > procs[i].burstTime) {
                    shortestBurst = procs[i].burstTime;
                    pick = i;
                }
            }
        }

        if (futureProc == 0 && pick == -1) // No more processes to be executed
            break;

        if (pick != -1) {
            gantt[timelapsed] = pick + 1;
            for (i=0; i<n; i++) {
                if (procs[i].burstTime <= 0) continue;
                if (i == pick)
                    procs[i].burstTime--;
                else
                    procs[i].waitTime++;
            }
        }
        else { // there will be some processes in future
            gantt[timelapsed] = -1;
        }

    } while (++timelapsed);

    printf("\n* Gantt Chart:\n");
    printf("0 ");
    for (i=1; i<=timelapsed; i++) {
        if (i % 5 == 0) printf("%2d", i);
        else printf("  ");
    }
    if (timelapsed % 5) printf("%2d", timelapsed);
```

```
190     printf("\n");
191     for (i=0; i<=timelapsed; i++) {
192         if (gantt[i] > 0) printf(" %d", gantt[i]);
193         else printf(" -");
194     }
195     printf("\n\n");
196 
197     double avgWait = 0.0;
198     for (i=0; i<n; i++) {
199         printf("Process %d waiting time = %d msec\n",
200                i + 1, procs[i].waitTime);
201         avgWait += procs[i].waitTime;
202     }
203     avgWait /= n;
204     printf("Average waiting time = %.1f msec\n", avgWait);
205 
206     return 0;
207 }
```