*/\**

*\* OS Assignment #2*

*\*/*

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <signal.h>

#include <ctype.h>

#include <errno.h>

#include <sys/wait.h>

#define MSG(x...) fprintf (stderr, x)

#define STRERROR strerror (errno)

#define PROCESS\_MAX (26 \* 10)

#define ID\_MAX 2

#define ARRIVE\_TIME\_MIN 0

#define ARRIVE\_TIME\_MAX 30

#define SERVICE\_TIME\_MIN 1

#define SERVICE\_TIME\_MAX 30

#define PRIORITY\_MIN 1

#define PRIORITY\_MAX 10

#define SLOT\_MAX ((ARRIVE\_TIME\_MAX + SERVICE\_TIME\_MAX) \* PROCESS\_MAX)

**enum**

{

SCHED\_SJF = 0,

SCHED\_SRT,

SCHED\_RR,

SCHED\_PR,

SCHED\_MAX

};

**typedef** **struct** \_Process Process;

**struct** \_Process

{

**int** idx;

**int** queue\_idx;

**char** id[ID\_MAX + 1];

**int** arrive\_time;

**int** service\_time;

**int** priority;

**int** remain\_time;

**int** complete\_time;

**int** turnaround\_time;

**int** wait\_time;

};

**static** Process processes[PROCESS\_MAX];

**static** **int** process\_total;

**static** Process \*queue[PROCESS\_MAX];

**static** **int** queue\_len;

**static** **char** schedule[PROCESS\_MAX][SLOT\_MAX];

**static** **char** \*strstrip (**char** \*str)

{

**char** \*start;

size\_t len;

len = strlen (str);

**while** (len--)

{

**if** (!isspace (str[len]))

**break**;

str[len] = '\0';

}

**for** (start = str; \*start && isspace (\*start); start++)

;

memmove (str, start, strlen (start) + 1);

**return** str;

}

**static** **int** check\_valid\_id (**const** **char** \*str)

{

size\_t len;

**int** i;

len = strlen (str);

**if** (len != ID\_MAX)

**return** -1;

**for** (i = 0; i < len; i++)

**if** (!(isupper (str[i]) || isdigit (str[i])))

**return** -1;

**return** 0;

}

**static** Process \*lookup\_process (**const** **char** \*id)

{

**int** i;

**for** (i = 0; i < process\_total; i++)

**if** (!strcmp (id, processes[i].id))

**return** &processes[i];

**return** **NULL**;

}

**static** **void** append\_process (Process \*process)

{

processes[process\_total] = \*process;

processes[process\_total].idx = process\_total;

process\_total++;

}

**static** **int** read\_config (**const** **char** \*filename)

{

FILE \*fp;

**char** line[256];

**int** line\_nr;

fp = fopen (filename, "r");

**if** (!fp)

**return** -1;

process\_total = 0;

line\_nr = 0;

**while** (fgets (line, **sizeof** (line), fp))

{

Process process;

**char** \*p;

**char** \*s;

size\_t len;

line\_nr++;

memset (&process, 0x00, **sizeof** (process));

len = strlen (line);

**if** (line[len - 1] == '\n')

line[len - 1] = '\0';

**if** (0)

MSG ("config[%3d] %s\n", line\_nr, line);

strstrip (line);

*/\* comment or empty line \*/*

**if** (line[0] == '#' || line[0] == '\0')

**continue**;

*/\* id \*/*

s = line;

p = strchr (s, ' ');

**if** (!p)

**goto** invalid\_line;

\*p = '\0';

strstrip (s);

**if** (check\_valid\_id (s))

{

MSG ("invalid process id '%s' in line %d, ignored\n", s, line\_nr);

**continue**;

}

**if** (lookup\_process (s))

{

MSG ("duplicate process id '%s' in line %d, ignored\n", s, line\_nr);

**continue**;

}

strcpy (process.id, s);

*/\* arrive time \*/*

s = p + 1;

p = strchr (s, ' ');

**if** (!p)

**goto** invalid\_line;

\*p = '\0';

strstrip (s);

process.arrive\_time = strtol (s, **NULL**, 10);

**if** (process.arrive\_time < ARRIVE\_TIME\_MIN

|| ARRIVE\_TIME\_MAX < process.arrive\_time

|| (process\_total > 0 &&

processes[process\_total - 1].arrive\_time > process.arrive\_time))

{

MSG ("invalid arrive-time '%s' in line %d, ignored\n", s, line\_nr);

**continue**;

}

*/\* service time \*/*

s = p + 1;

p = strchr (s, ' ');

**if** (!p)

**goto** invalid\_line;

\*p = '\0';

strstrip (s);

process.service\_time = strtol (s, **NULL**, 10);

**if** (process.service\_time < SERVICE\_TIME\_MIN

|| SERVICE\_TIME\_MAX < process.service\_time)

{

MSG ("invalid service-time '%s' in line %d, ignored\n", s, line\_nr);

**continue**;

}

*/\* priority \*/*

s = p + 1;

strstrip (s);

process.priority = strtol (s, **NULL**, 10);

**if** (process.priority < PRIORITY\_MIN

|| PRIORITY\_MAX < process.priority)

{

MSG ("invalid priority '%s' in line %d, ignored\n", s, line\_nr);

**continue**;

}

append\_process (&process);

**continue**;

invalid\_line:

MSG ("invalid format in line %d, ignored\n", line\_nr);

}

fclose (fp);

**return** 0;

}

**static** **void** simulate (**int** sched)

{

Process \*process;

**int** p;

**int** p\_done;

**int** cpu\_time;

**int** sum\_turnaround\_time;

**int** sum\_waiting\_time;

**float** avg\_turnaround\_time;

**float** avg\_waiting\_time;

**for** (p = 0; p < PROCESS\_MAX; p++)

{

**int** slot;

**for** (slot = 0; slot < SLOT\_MAX; slot++)

schedule[p][slot] = 0;

queue[p] = **NULL**;

}

p = 0;

p\_done = 0;

queue\_len = 0;

process = **NULL**;

**for** (cpu\_time = 0; p\_done < process\_total; cpu\_time++)

{

*/\* Insert arrived process into the queue. \*/*

**for** (; p < process\_total; p++)

{

Process \*pp;

pp = &processes[p];

**if** (pp->arrive\_time != cpu\_time)

**break**;

pp->remain\_time = pp->service\_time;

pp->queue\_idx = queue\_len;

queue[queue\_len] = pp;

queue\_len++;

}

*/\* Pick a process according to scheduling algorithm. \*/*

**switch** (sched)

{

**case** SCHED\_SJF:

**if** (!process)

{

**int** i;

**int** shortest;

shortest = SERVICE\_TIME\_MAX + 1;

**for** (i = 0; i < queue\_len; i++)

**if** (queue[i]->service\_time < shortest)

{

process = queue[i];

shortest = process->service\_time;

}

}

**break**;

**case** SCHED\_SRT:

{

**int** i;

**int** shortest;

shortest = SERVICE\_TIME\_MAX + 1;

**for** (i = 0; i < queue\_len; i++)

**if** (queue[i]->remain\_time < shortest)

{

process = queue[i];

shortest = process->remain\_time;

}

}

**break**;

**case** SCHED\_RR:

{

**int** i;

process = queue[0];

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

{

queue[i] = queue[i + 1];

queue[i]->queue\_idx = i;

}

queue[i] = process;

queue[i]->queue\_idx = i;

}

**break**;

**case** SCHED\_PR:*//*추가된 *PR* 스케줄링알고리즘

{

**int** i;

**int** temp\_pr; *//temporary priority*를저장할변수

temp\_pr = PRIORITY\_MAX+1;

**for**(i = 0;i < queue\_len;i++)

{

**if**(queue[i] -> priority < temp\_pr)

{

process = queue[i];

temp\_pr = process -> priority;

}

}

}

**break**;

**default**:

MSG ("invalid scheduing algorithm '%d', ignored\n", sched);

**return**;

}

**if** (0)

MSG ("[%02d] %s[%d:%d] %d/%d\n",

cpu\_time,

process->id,

process->idx,

process->queue\_idx,

process->remain\_time,

process->service\_time);

*/\* no process to schedule. \*/*

**if** (!process)

**continue**;

schedule[process->idx][cpu\_time] = 1;

process->remain\_time--;

**if** (process->remain\_time <= 0)

{

**int** i;

**for** (i = process->queue\_idx; i < (queue\_len - 1); i++)

{

queue[i] = queue[i + 1];

queue[i]->queue\_idx = i;

}

queue\_len--;

process->complete\_time = cpu\_time + 1;

process->turnaround\_time =

process->complete\_time - process->arrive\_time;

process->wait\_time =

process->turnaround\_time - process->service\_time;

process = **NULL**;

p\_done++;

}

}

printf ("\n[%s]\n",

sched == SCHED\_SJF ? "SJF" :

sched == SCHED\_SRT ? "SRT" :

sched == SCHED\_RR ? "RR" :

sched == SCHED\_PR ? "PR": "UNKNOWN");

sum\_turnaround\_time = 0;

sum\_waiting\_time = 0;

**for** (p = 0; p < process\_total; p++)

{

**int** slot;

*//*간트챠트형식의출력

printf ("%s ", processes[p].id);

**for** (slot = 0; slot <= cpu\_time; slot++)

putchar (schedule[p][slot] ? '\*' : ' ');

printf ("\n");

sum\_turnaround\_time += processes[p].turnaround\_time;

sum\_waiting\_time += processes[p].wait\_time;

}

avg\_turnaround\_time = (**float**) sum\_turnaround\_time / (**float**) process\_total;

avg\_waiting\_time = (**float**) sum\_waiting\_time / (**float**) process\_total;

printf ("CPU TIME: %d\n", cpu\_time);

printf ("AVERAGE TURNAROUND TIME: %.2f\n", avg\_turnaround\_time);

printf ("AVERAGE WAITING TIME: %.2f\n", avg\_waiting\_time);

}

**int** main (**int** argc, **char** \*\*argv)

{

**int** sched;

**if** (argc <= 1)

{

MSG ("usage: %s input-file\n", argv[0]);

**return** -1;

}

**if** (read\_config (argv[1]))

{

MSG ("failed to load config file '%s': %s\n", argv[1], STRERROR);

**return** -1;

}

**for** (sched = 0; sched < SCHED\_MAX; sched++)

simulate (sched);

**return** 0;

}