106119029 , OS Lab 6

Dipesh Kafle

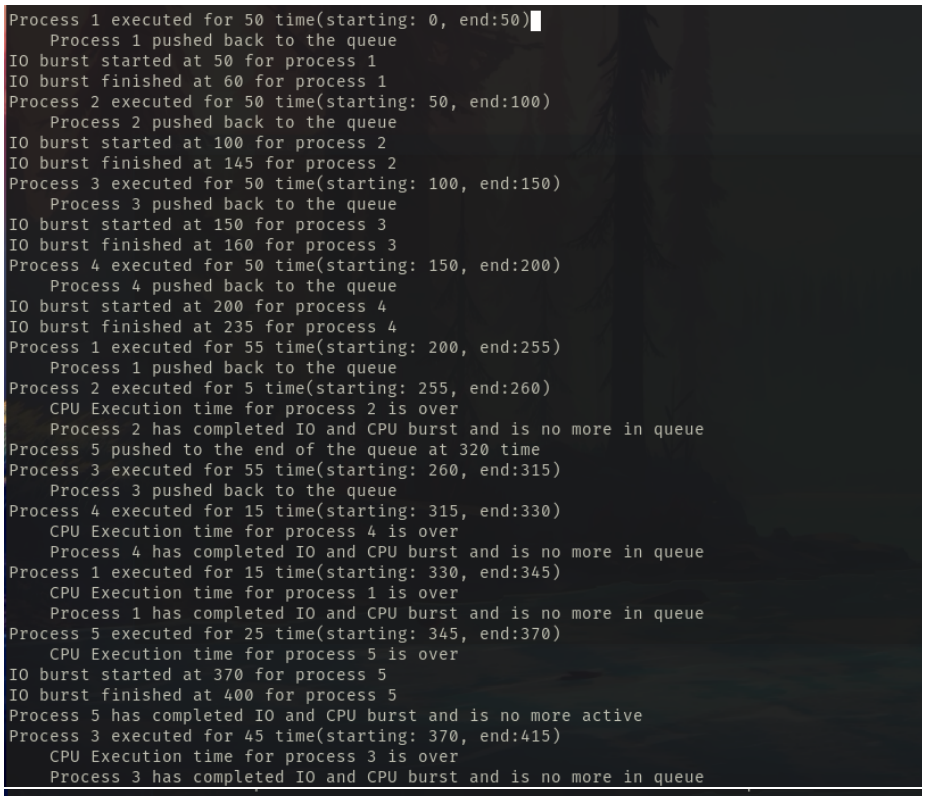
Code

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>

#define min(x, y) x > y ? y : x

pthread_mutex_t lock;

struct process_detail {
  u_int32_t id;
  u_int32_t arrival;
  u_int32_t cpu_burst;
  u_int32_t io_burst;
  u_int32_t priority;
  u_int32_t time_quanta;
  u_int32_t finish_time;
};

struct process_detail *queue[1000];

struct process_detail_for_io_exec {
  struct process_detail *process;
  u_int32_t start_time;
};

struct process_add_timer {
  struct process_detail *process;
  u_int32_t *r;
};
```

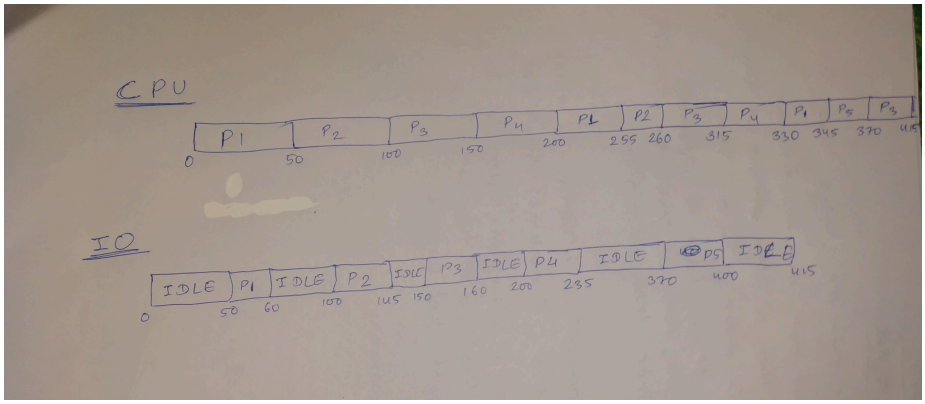
```c
void *put_in_queue(void *data) {
  struct process_add_timer *p = (struct process_add_timer *)data;
  usleep(p->process->arrival * 100);
  pthread_mutex_lock(&lock);
  queue[(*(p->r))++] = p->process;

  pthread_mutex_unlock(&lock);
  printf("Process %d pushed to the end of the queue at %d time \n",
         p->process->id, p->process->arrival);
  fflush(stdout);
}

void *sleep_func(void *param) {
  struct process_detail_for_io_exec p;
  memcpy(&p, param, sizeof(struct process_detail_for_io_exec));
  u_int32_t runtime = min(p.process->io_burst, p.process->time_quanta);
  printf("IO burst started at %d for process %d\n", p.start_time,
         p.process->id);
  fflush(stdout);
  usleep(p.process->io_burst * 100);
  printf("IO burst finished at %d for process %d\n", p.start_time + runtime,
         p.process->id);
  fflush(stdout);
  p.process->io_burst -= runtime;
  p.process->time_quanta -= 5;
  if (p.process->cpu_burst == 0 && p.process->io_burst == 0) {
    printf("Process %d has completed IO and CPU burst and is no more active\n",
           p.process->id);
    p.process->finish_time = p.start_time + runtime;
  }
  return NULL;
}
```

```c
int main() {
  pthread_t threads[100];
  int thread_count = 0;

  struct process_detail p1 = {1, 0, 120, 10, 5, 50, 0};
  struct process_detail p2 = {2, 0, 55, 45, 5, 50, 0};
  struct process_detail p3 = {3, 0, 150, 10, 5, 50, 0};
  struct process_detail p4 = {4, 0, 65, 35, 5, 50, 0};
  struct process_detail p5 = {5, 320, 25, 30, 5, 50, 0};

  queue[0] = &p1;
  queue[1] = &p2;
  queue[2] = &p3;
  queue[3] = &p4;
  u_int32_t l = 0;
  u_int32_t r = 4;
  u_int32_t seconds = 0;
  u_int32_t last_stop = 0;

  struct process_add_timer p = {&p5, &r};
  // create a thread which will put process 5 to the queue
  pthread_create(threads + (thread_count++), NULL, put_in_queue, &p);

  int io = 0;
  while (r != l) {

    u_int32_t runtime = min(queue[l]->cpu_burst, queue[l]->time_quanta);

    seconds += runtime;
    usleep(runtime * 100);
    queue[l]->time_quanta =
        min(queue[l]->time_quanta + (runtime == queue[l]->time_quanta ? 10 : 0),
            100);
    queue[l]->cpu_burst -= runtime;
    queue[l]->priority++;
    printf("Process %d executed for %d time(starting: %d, end:%d)\n ",
           queue[l]->id, runtime, seconds - runtime, seconds);
    fflush(stdout);

    if (queue[l]->io_burst) {
      struct process_detail_for_io_exec data = {queue[l], seconds};
      pthread_create(threads + thread_count, NULL, sleep_func, &data);
      thread_count++;
    }

    if (queue[l]->cpu_burst) {
      printf("   Process %d pushed back to the queue\n", queue[l]->id);
      queue[r++] = queue[l++];
    } else {
      printf("   CPU Execution time for process %d is over\n", queue[l]->id);
      // if io_burst and cpu_burst both completed process is done
      if (queue[l]->io_burst == 0) {
        printf(
            "    Process %d has completed IO and CPU burst and is no more in "
            "queue\n",
            queue[l]->id);
        queue[l]->finish_time = seconds;
      }
      l++;
    }
    fflush(stdout);
  }

  printf("\n\n\n");
  struct process_detail *all_processes[5] = {&p1, &p2, &p3, &p4, &p5};

  double turnaroundtime = 0;
  for (int i = 0; i < 5; i++) {
    printf("Process %d arrived at %d and finished at %d\n",
           all_processes[i]->id, all_processes[i]->arrival,
           all_processes[i]->finish_time);
    turnaroundtime +=
        (all_processes[i]->finish_time - all_processes[i]->arrival);
  }
  printf("\nThe average turnaroundtime is %f\n", turnaroundtime / 5);
}
```

Output


Process 1 executed for 50 time(starting: 0, end:50)
Process 1 pushed back to the queue
IO burst started at 50 for process 1
IO burst finished at 60 for process 1
Process 2 executed for 50 time(starting: 50, end:100)
Process 2 pushed back to the queue
IO burst started at 100 for process 2
IO burst finished at 145 for process 2
Process 3 executed for 50 time(starting: 100, end:150)
Process 3 pushed back to the queue
IO burst started at 150 for process 3
IO burst finished at 160 for process 3
Process 4 executed for 50 time(starting: 150, end:200)
Process 4 pushed back to the queue
IO burst started at 200 for process 4
IO burst finished at 235 for process 4
Process 1 executed for 55 time(starting: 200, end:255)
Process 1 pushed back to the queue
Process 2 executed for 5 time(starting: 255, end:260)
CPU Execution time for process 2 is over
Process 2 has completed IO and CPU burst and is no more in queue
Process 5 pushed to the end of the queue at 320 time
Process 3 executed for 55 time(starting: 260, end:315)
Process 3 pushed back to the queue
Process 4 executed for 15 time(starting: 315, end:330)
CPU Execution time for process 4 is over
Process 4 has completed IO and CPU burst and is no more in queue
Process 1 executed for 15 time(starting: 330, end:345)
CPU Execution time for process 1 is over
Process 1 has completed IO and CPU burst and is no more in queue
Process 5 executed for 25 time(starting: 345, end:370)
CPU Execution time for process 5 is over
IO burst started at 370 for process 5
IO burst finished at 400 for process 5
Process 5 has completed IO and CPU burst and is no more active
Process 3 executed for 45 time(starting: 370, end:415)
CPU Execution time for process 3 is over
Process 3 has completed IO and CPU burst and is no more in queue
Process 1 arrived at 0 and finished at 345
Process 2 arrived at 0 and finished at 260
Process 3 arrived at 0 and finished at 415
Process 4 arrived at 0 and finished at 330
Process 5 arrived at 320 and finished at 400
The average turnaroundtime is 286.000000

Gantt Chart



- I have a queue for CPU and I've simulated IO using threads. After one quanta is over for a process, it will wait for IO in a separate thread for the io_burst amount of time. That way I have simulated IO and CPU times.
- p1,p2,p3 and p4 are put into the queue at 0 time.
- At 0, P1 starts the execution and continues till 50 time units. Then it will stop there and add 10 time units to its quanta because it executed for a whole quanta without waiting for IO. It will also spawn a thread for IO waiting, which will sleep for 10 time units and terminate after modifying the values and printing. `At t=50, p1.time_quanta=60`.
- At 50-60 time interval in the IO thread, P1 has finished its IO and printed. Since IO was done, time quanta of P1 is reduced. `At t=60, p1.time_quanta = 60-5=55` .
- At t=50, P2 starts its CPU execution and continues until 100 in the CPU ready queue. It then spawns a thread to do IO (which goes on for 45 seconds from 100 to 145 time units). So for the next CPU cycle of P2, it will have `55 time units` as it did one whole quanta of CPU bound work and also did IO work. So `50+10-5`.
- At t=100, P3 starts executing till 150 and spawns a thread for IO which will go on for 10 time units (till 160). It's next time quanta will also be `55 time units`.
- At t=150, P4 starts off and goes on for a quanta till 200 and then spawns a thread which goes on for 35 time units. It's time quanta for next cycle will also be `55 time units`.
- At t=200, P1 is scheduled and it starts off and works for 55 time units(one full quanta) and since IO is over for P1, it wont spawn a new thread. It's time quanta is increased to `65 time units`.

- At t=255 , P2 starts off and goes on till 260 and finishes all its work and prints that. Hands the control to scheduler at 260.
- At t=260, P3 starts and goes on for 55 time units(its time quanta) and is pushed back to the queue again. No IO is done as its IO time is over already.
- At t=315, P4 starts off and goes on for 15 time units until it completes all its work. Wont be available for scheduling again.
- At t=330, P1 is given the CPU and it will complete all its work at 345 and hand the control to the scheduler.
- At t=345, P5 starts the execution for the first time ever and finishes CPU work by 370. Spawns a new thread at t=370 and hands the control to scheduler. The IO will complete at t=400, after which process P5 will have finished all its work.
- At t=370, CPU is handed over to P3 which will complete everything by 415 and now all our processes have been completed.
- From the output we can see that the Average Turnaround time is 286 time units.

Remarks

- It is obvious by how the priority is changing this scheudling method will favor a more CPU bounded process.

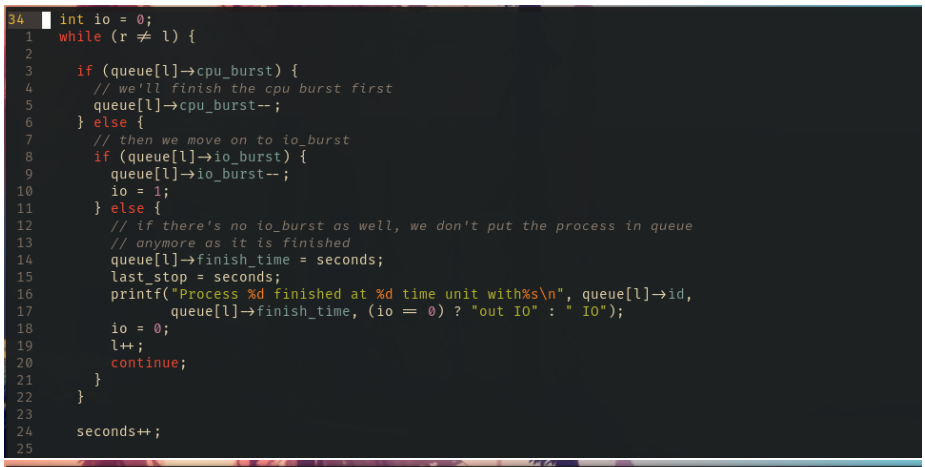
Alternative approach for the sake of comparision

- I have also made another scheduler, A classic round robin scheduler. In that, CPU time and IO time are treated as same for the sake of simplicity. When I schedule these processes with that scheduler, the processes are finishing at `545 time units` which is much higher. Obviously the turnaround time is more for it (387 time units).
- Output for normal RR scheduler

```
Process 1 executed for one time quanta of 50 time unit(without IO), pushed back to the end of queue at 50
Process 2 executed for one time quanta of 50 time unit(without IO), pushed back to the end of queue at 100
Process 3 executed for one time quanta of 50 time unit(without IO), pushed back to the end of queue at 150
Process 4 executed for one time quanta of 50 time unit(without IO), pushed back to the end of queue at 200
Process 1 executed for one time quanta of 60 time unit(without IO), pushed back to the end of queue at 260
Process 2 finished at 310 time unit with IO
Process 5 pushed to the end of queue at 320
Process 3 executed for one time quanta of 60 time unit(without IO), pushed back to the end of queue at 370
Process 4 finished at 420 time unit with IO
Process 1 finished at 440 time unit with IO
Process 5 executed for one time quanta of 50 time unit(with IO), pushed back to the end of queue at 490
Process 3 finished at 540 time unit with IO
Process 5 finished at 545 time unit with IO



Process 1 arrived at 0 and finished at 440
Process 2 arrived at 0 and finished at 310
Process 3 arrived at 0 and finished at 540
Process 4 arrived at 0 and finished at 420
Process 5 arrived at 320 and finished at 545

The average turnaroundtime is 387.000000
```

- Code for normal RR scheduler

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

struct process_detail
{
  u_int32_t id;
  u_int32_t arrival;
  u_int32_t cpu_burst;
  u_int32_t io_burst;
  u_int32_t priority;
  u_int32_t time_quanta;
  u_int32_t finish_time;
};

int main() {
  struct process_detail p1 = {1, 0, 120, 10, 5, 50, 0};
  struct process_detail p2 = {2, 0, 55, 45, 5, 50, 0};
  struct process_detail p3 = {3, 0, 150, 10, 5, 50, 0};
  struct process_detail p4 = {4, 0, 65, 35, 5, 50, 0};
  struct process_detail p5 = {5, 320, 25, 30, 5, 50, 0};

  struct process_detail *queue[1000] = {NULL};
  queue[0] = &p1;
  queue[1] = &p2;
  queue[2] = &p3;
  queue[3] = &p4;
  int l = 0;
  int r = 4;
  /* u_int32_t time_quanta = 50; */
  u_int32_t seconds = 0;
  u_int32_t last_stop = 0;

```



```c
int io = 0;
while (r != l) {

  if (queue[l]->cpu_burst) {
    // we'll finish the cpu burst first
    queue[l]->cpu_burst--;
  } else {
    // then we move on to io_burst
    if (queue[l]->io_burst) {
      queue[l]->io_burst--;
      io = 1;
    } else {
      // if there's no io_burst as well, we don't put the process in queue
      // anymore as it is finished
      queue[l]->finish_time = seconds;
      last_stop = seconds;
      printf("Process %d finished at %d time unit with%s\n", queue[l]->id,
             queue[l]->finish_time, (io == 0) ? "out IO" : " IO");
      io = 0;
      l++;
      continue;
    }
  }

  seconds++;
```

```c
  if (seconds == 320) {
    queue[r++] = &p5;
    printf("Process %d pushed to the end of queue at %d\n", queue[r - 1]->id,
           queue[r - 1]->arrival);
  }
  // if the time quanta is finished
  if (seconds - last_stop == queue[l]->time_quanta) {
    printf("Process %d executed for one time quanta of %d time unit(with%s), "
           "pushed "
           "back to the end of queue at %d\n",
           queue[l]->id, queue[l]->time_quanta, (io == 0) ? "out IO" : " IO",
           seconds);
    if (io) {
      queue[l]->time_quanta -= 5;
    } else {
      queue[l]->time_quanta += (queue[l]->time_quanta == 100 ? 0 : 10);
    }
    queue[r++] = queue[l++];
    last_stop = seconds;
  }
}
printf("\n\n\n");
struct process_detail *all_processes[5] = {&p1, &p2, &p3, &p4, &p5};

double turnaroundtime = 0;
for (int i = 0; i < 5; i++) {
  printf("Process %d arrived at %d and finished at %d\n",
         all_processes[i]->id, all_processes[i]->arrival,
         all_processes[i]->finish_time);
  turnaroundtime +=
      (all_processes[i]->finish_time - all_processes[i]->arrival);
}
printf("\nThe average turnaroundtime is %f\n", turnaroundtime / 5);
}
```