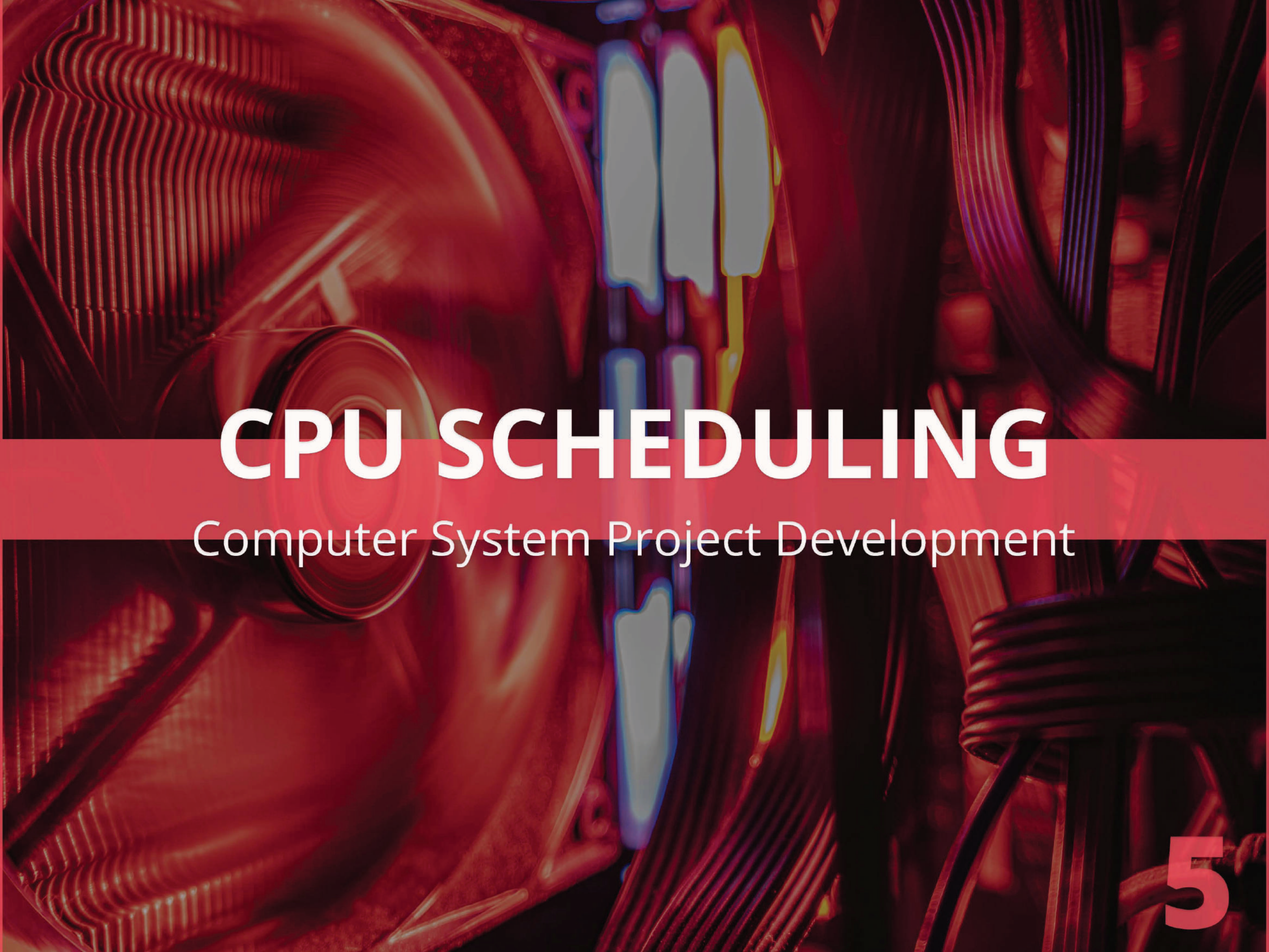


CPU SCHEDULING

Computer System Project Development

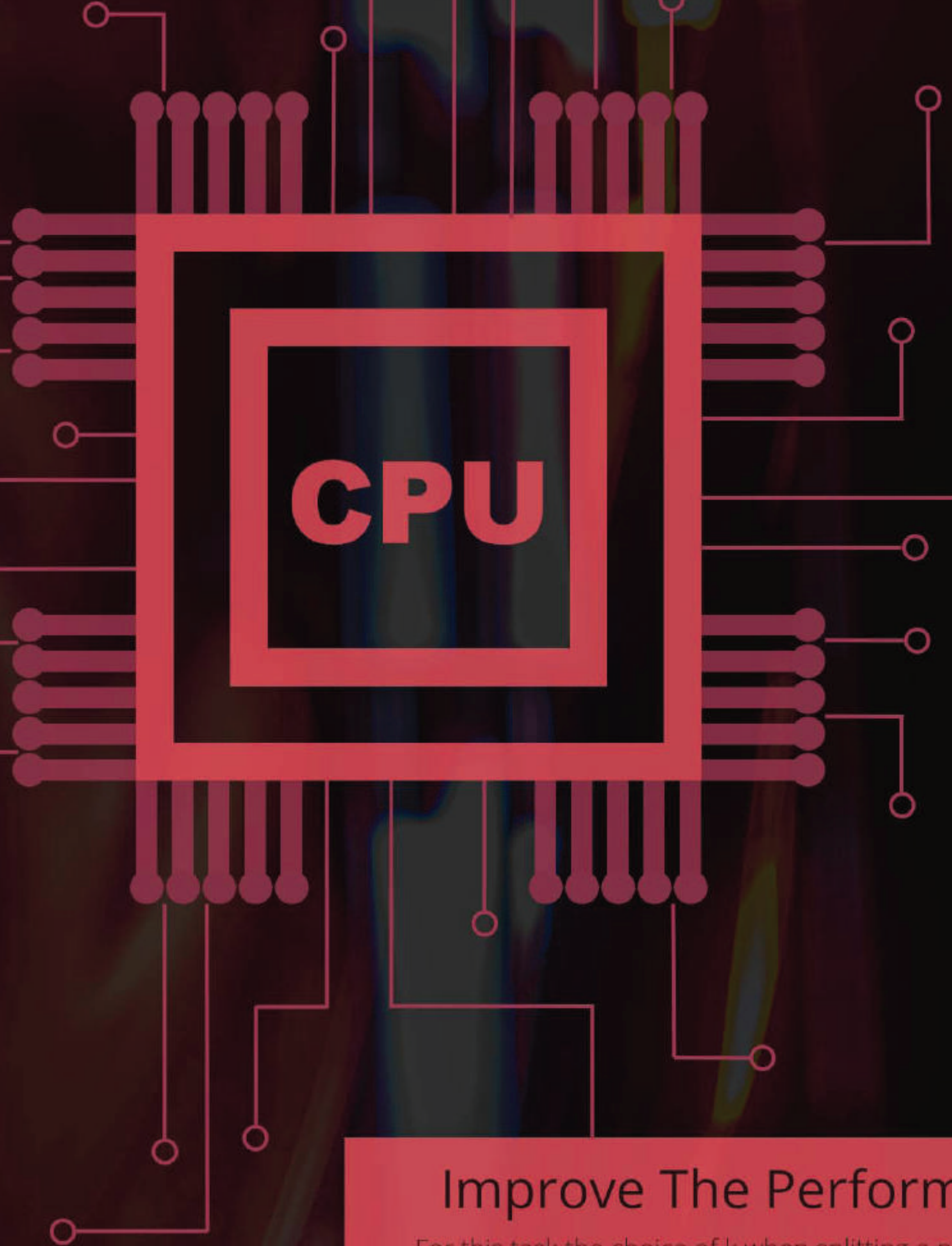
5

Introduction

This program will be invoked via the command line. It will be given a description of arriving processes including their arrival time, execution time in seconds, their id, and whether they are parallelisable or not. The first process always arrives at time 0, which is when the simulation begins.

Single Processor Scheduler

Processes will be allocated CPU time according to shortest-time-remaining algorithm, a preemptive scheduling algorithm. When a new process arrives, it is scheduled if there is no other process running. If, when process j arrives, there is a process i running, then i is postponed and added to a queue if and only if j has a shorter execution time than the remaining time of i. Process i is resumed where it left off, if it is the process with the shortest remaining time left among all other processes in the queue (i.e., including those that may have arrived after j). Process ids are used to break ties, giving preference to processes with smaller ids. The simulation should terminate once all processes have run to completion.



Two Processor Schedule

In this section, scheduler will be extended to work with 2 processors: processors numbered 0 and 1. This will simulate a situation where processes can run truly in parallel and a process that can run faster with more computational resources (e.g., by creating child/sub processes). If the process that arrives is not parallelisable, it is assigned to the queue of the CPU with the smallest amount of remaining execution time for all processes and subprocesses (defined below) allocated to it, using CPU id to break ties, giving preference to CPU 0. After that the simulation for this process behaves same as in Section 1. In this project, once a process is assigned to a CPU it cannot be moved to another CPU. (Think of why it may not be a good idea to migrate processes between CPUs.) If arriving process with id i and execution time x is parallelisable, two subprocesses are created, i.0 and i.1, each with execution time dx/2e + 1. The extra time simulates the cost of synchronisation. For example, a parallelisable process that runs on a single CPU in 6 seconds, can finish within 4 seconds if both CPUs are idle when it arrives. Once subprocesses are created, subprocess i.0 is added to the queue of CPU 0 and i.1 is added to CPU 1. After that they are treated as regular processes when scheduling them on each CPU. A parallelisable process is considered finished when both of its subprocesses have finished.

N Processor Scheduler

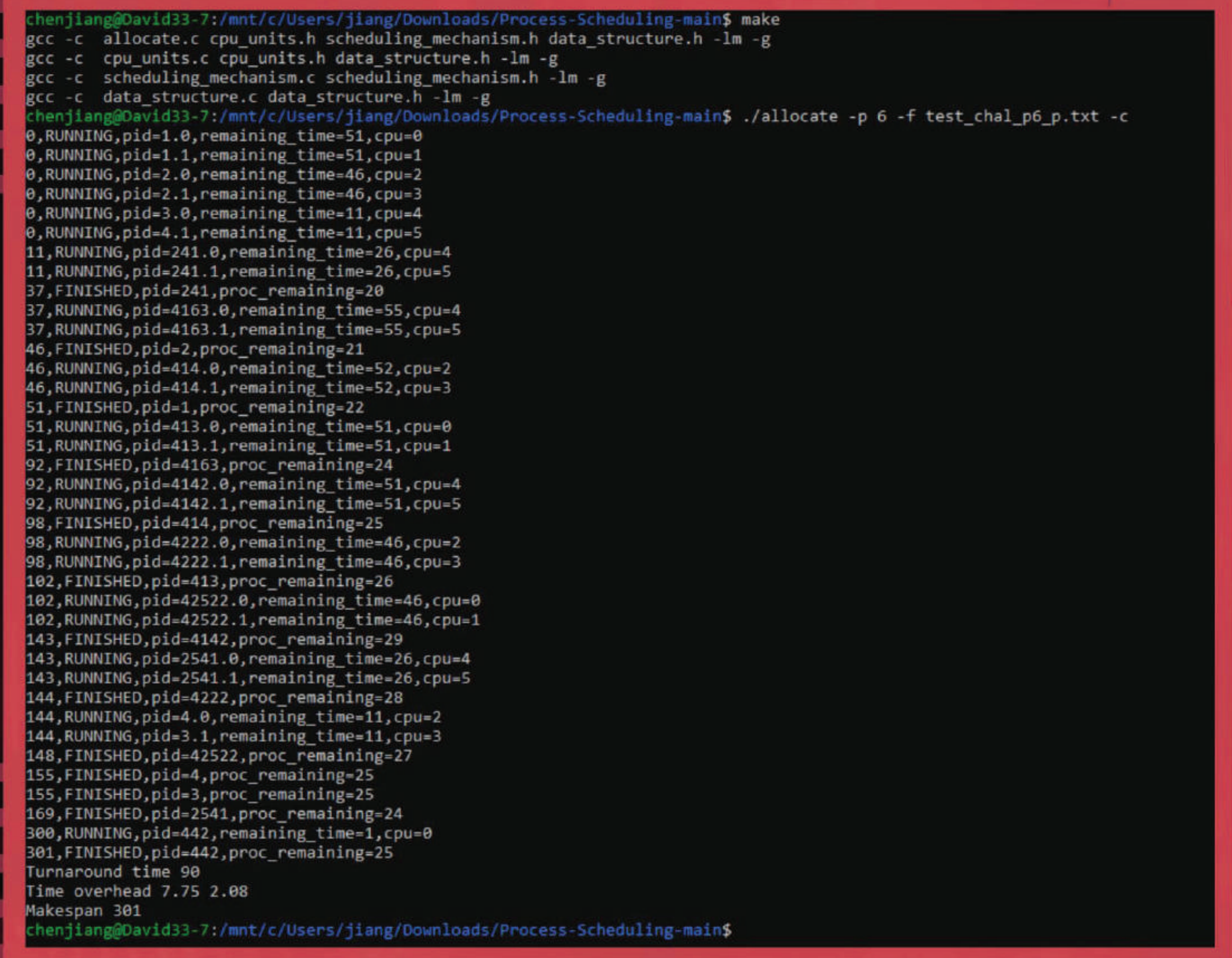
2 processors setting will be generalised to N ≥ 3 processors. Similar to above, a non-parallelisable process is assigned to a CPU that has the shortest remaining time to complete all processes and subprocesses assigned to it so far.
A parallelisable process is split into k ≤ N subprocesses, where k is the largest value for which x/k ≥ 1. Each subprocess is then assigned execution time of [x/k] + 1. Subprocesses follow a similar naming convention as above: a process with id i is split into subprocesses with id's i.0, i.1, . . ., i.k′, where k′ = k − 1. Subprocesses are then assigned to k processors with shortest time left to complete such that subprocess i.0 is assigned to the CPU that can finish its processes the fastest, i.1 is assigned to the second fastest processor and so on. Use CPU ids to break ties, giving preference to smaller processor numbers. Once a process or subprocess is assigned to a CPU it cannot migrate to another CPU. A parallelisable process is considered finished when all of its subprocesses have finished.

Improve The Performance

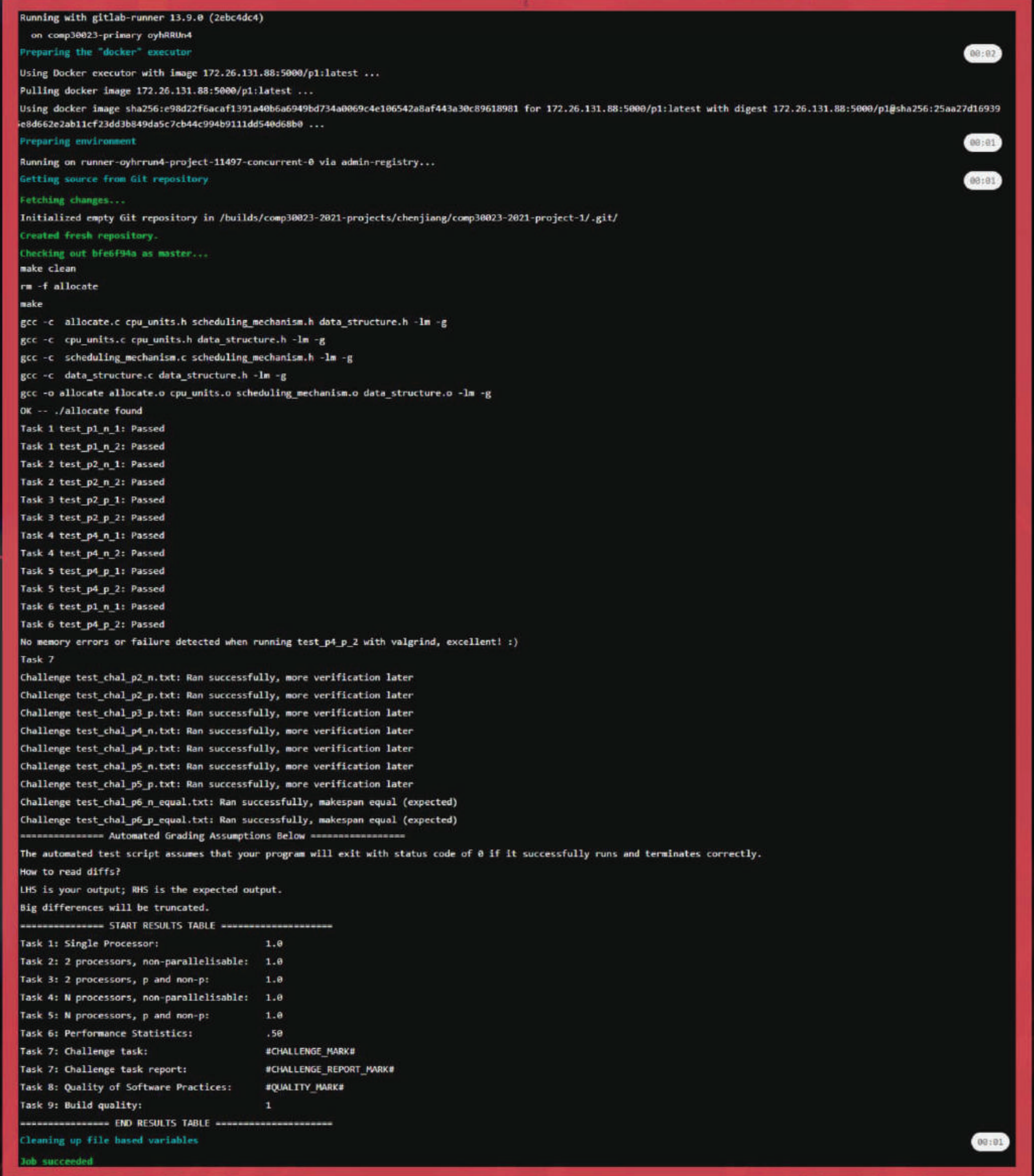
For this task the choice of k when splitting a parallelisable process is adjusted. The program will be allowed to "look into the future" and see what processes will be arriving. Therefore, All processes are assigned to Waiting Area first. Once a CPU idle, the longest process will be pushed to the idle-CPU from the waiting Area.

OUTCOME

CODE: github.com/chenjiang0819/Process-Scheduling



```
chenjiang@David33-7:/mnt/c/Users/jiang/Downloads/Process-Scheduling-main$ make
gcc -c  allocate.c cpu_units.h scheduling_mechanism.h data_structure.h -lm -g
gcc -c  cpu_units.c cpu_units.h data_structure.h -lm -g
gcc -c  scheduling_mechanism.c scheduling_mechanism.h -lm -g
gcc -c  data_structure.c data_structure.h -lm -g
chenjiang@David33-7:/mnt/c/Users/jiang/Downloads/Process-Scheduling-main$ ./allocate -p 6 -f test_chal_p6_p.txt -c
0,RUNNING,pid=1.0,remaining_time=51,cpu=0
0,RUNNING,pid=1.1,remaining_time=51,cpu=1
0,RUNNING,pid=2.0,remaining_time=46,cpu=2
0,RUNNING,pid=2.1,remaining_time=46,cpu=3
0,RUNNING,pid=3.0,remaining_time=11,cpu=4
0,RUNNING,pid=4.1,remaining_time=11,cpu=5
11,RUNNING,pid=241.0,remaining_time=26,cpu=4
11,RUNNING,pid=241.1,remaining_time=26,cpu=5
37,FINISHED,pid=241,proc_remaining=20
37,RUNNING,pid=4163.0,remaining_time=55,cpu=4
37,RUNNING,pid=4163.1,remaining_time=55,cpu=5
46,FINISHED,pid=2,proc_remaining=21
46,RUNNING,pid=414.0,remaining_time=52,cpu=2
46,RUNNING,pid=414.1,remaining_time=52,cpu=3
51,FINISHED,pid=1,proc_remaining=22
51,RUNNING,pid=413.0,remaining_time=51,cpu=0
51,RUNNING,pid=413.1,remaining_time=51,cpu=1
92,FINISHED,pid=4163,proc_remaining=24
92,RUNNING,pid=4142.0,remaining_time=51,cpu=4
92,RUNNING,pid=4142.1,remaining_time=51,cpu=5
98,FINISHED,pid=414,proc_remaining=25
98,RUNNING,pid=4222.0,remaining_time=46,cpu=2
98,RUNNING,pid=4222.1,remaining_time=46,cpu=3
102,FINISHED,pid=413,proc_remaining=26
102,RUNNING,pid=42522.0,remaining_time=46,cpu=0
102,RUNNING,pid=42522.1,remaining_time=46,cpu=1
143,FINISHED,pid=4142,proc_remaining=29
143,RUNNING,pid=2541.0,remaining_time=26,cpu=4
143,RUNNING,pid=2541.1,remaining_time=26,cpu=5
144,FINISHED,pid=4222,proc_remaining=28
144,RUNNING,pid=4.0,remaining_time=11,cpu=2
144,RUNNING,pid=3.1,remaining_time=11,cpu=3
148,FINISHED,pid=42522,proc_remaining=27
155,FINISHED,pid=4,proc_remaining=25
155,FINISHED,pid=3,proc_remaining=25
169,FINISHED,pid=2541,proc_remaining=24
300,RUNNING,pid=442,remaining_time=1,cpu=0
301,FINISHED,pid=442,proc_remaining=25
Turnaround time 90
Time overhead 7.75 2.08
Makespan 301
chenjiang@David33-7:/mnt/c/Users/jiang/Downloads/Process-Scheduling-main$
```



```
Running with gitlab-runner 13.9.0 (2ebc4dc4)
  on comp30023-primary oyhRRUn4
Preparing the "docker" executor                                                    00:02
Using Docker executor with image 172.26.131.88:5000/p1:latest ...
Pulling docker image 172.26.131.88:5000/p1:latest ...
Using docker image sha256:e98d22f6acaf1391a40b6a6949bd734a0069c4e106542a8af443a30c89618981 for 172.26.131.88:5000/p1:latest with digest 172.26.131.88:5000/p1@sha256:25aa27d16939
e8d662e2ab11cf23dd3b849da5c7cb44c994b9111dd540d68b0 ...
Preparing environment                                                              00:01
Running on runner-oyhrrun4-project-11497-concurrent-0 via admin-registry...
Getting source from Git repository                                                 00:01
Fetching changes...
Initialized empty Git repository in /builds/comp30023-2021-projects/chenjiang/comp30023-2021-project-1/.git/
Created fresh repository.
Checking out bfe6f94a as master...
make clean
rm -f allocate
make
gcc -c  allocate.c cpu_units.h scheduling_mechanism.h data_structure.h -lm -g
gcc -c  cpu_units.c cpu_units.h data_structure.h -lm -g
gcc -c  scheduling_mechanism.c scheduling_mechanism.h -lm -g
gcc -c  data_structure.c data_structure.h -lm -g
gcc -o allocate allocate.o cpu_units.o scheduling_mechanism.o data_structure.o -lm -g
OK -- ./allocate found
Task 1 test_p1_n_1: Passed
Task 1 test_p1_n_2: Passed
Task 2 test_p2_n_1: Passed
Task 2 test_p2_n_2: Passed
Task 3 test_p2_p_1: Passed
Task 3 test_p2_p_2: Passed
Task 4 test_p4_n_1: Passed
Task 4 test_p4_n_2: Passed
Task 5 test_p4_p_1: Passed
Task 5 test_p4_p_2: Passed
Task 6 test_p1_n_1: Passed
Task 6 test_p4_p_2: Passed
No memory errors or failure detected when running test_p4_p_2 with valgrind, excellent! :)
Task 7
Challenge test_chal_p2_n.txt: Ran successfully, more verification later
Challenge test_chal_p2_p.txt: Ran successfully, more verification later
Challenge test_chal_p3_p.txt: Ran successfully, more verification later
Challenge test_chal_p4_n.txt: Ran successfully, more verification later
Challenge test_chal_p4_p.txt: Ran successfully, more verification later
Challenge test_chal_p5_n.txt: Ran successfully, more verification later
Challenge test_chal_p5_p.txt: Ran successfully, more verification later
Challenge test_chal_p6_n_equal.txt: Ran successfully, makespan equal (expected)
Challenge test_chal_p6_p_equal.txt: Ran successfully, makespan equal (expected)
=============== Automated Grading Assumptions Below =================
The automated test script assumes that your program will exit with status code of 0 if it successfully runs and terminates correctly.
How to read diffs?
LHS is your output; RHS is the expected output.
Big differences will be truncated.
=============== START RESULTS TABLE ====================
Task 1: Single Processor:                  1.0
Task 2: 2 processors, non-parallelisable:  1.0
Task 3: 2 processors, p and non-p:         1.0
Task 4: N processors, non-parallelisable:  1.0
Task 5: N processors, p and non-p:         1.0
Task 6: Performance Statistics:            .50
Task 7: Challenge task:                    #CHALLENGE_MARK#
Task 7: Challenge task report:             #CHALLENGE_REPORT_MARK#
Task 8: Quality of Software Practices:     #QUALITY_MARK#
Task 9: Build quality:                     1
================ END RESULTS TABLE =====================
Cleaning up file based variables                                                   00:01
Job succeeded
```