2017-18 Programming Project

COMP3230A Principles of Operating Systems

Task – Compute-Intensive Application

- You are going to implement two C programs – the first one is a multiprocessing program and the other is a multithreaded program – to make use of multiple CPU cores to speed-up the computation in generating the Mandelbrot set image

- This is an individual project; not a group project.

Objectives

- An assessment task related to ILO4 [Practicability]
- A learning activity related to ILO 2a and ILO 2c
- The goals of this programming project are:
  - to design and develop multiprocessing and multithreaded programs
  - to learn how to use various Unix system functions in process creation and interaction
  - to learn how to use POSIX pthreads library in thread creation and coordination
  - to implement the bounded-buffer problem.

Computer platform to use

- Your programs must be written in C and successfully compiled with gcc
- You are expected to develop and test your programs under Ubuntu 16.04
  - Using the Ubuntu 16.04 VM provided by the course; **Or**
  - Install a native Ubuntu installation on your notebook/desktop; however, you have to install the SDL2 graphic library
- Do the final test on the department's Linux virtual machine: workbench2.cs.hku.hk
  - A virtual Linux server dedicated for the OS course and has installed with SDL2 library
  - This virtual server is equipped (or allocated) with 40 logical cores, so you can test your programs with more threads or processes

Accessing workbench2.cs.hku.hk

Via X2Go client

- Connect to HKUCS VPN
- Run the X2Go client and connect to **academy.cs.hku.hk**
- Open a terminal
- Execute the following command for connecting to workbench2
  - ssh **-X** workbench2

Via Course's Ubuntu VM

- Connect to HKUCS VPN
  - Setting this in the VM or in the host environment
- Open a terminal (in the VM)
- Execute the following command
  - ssh **-l** "CS_username" **-X** workbench2.cs.hku.hk

Structure of the Project

- Given: Sequential Program
  - Provide the logics on image drawing and Mandelbrot set computation
- Part One [9 points]
  - Stage A: Simple Multiprocessing Program
    - + Process Management
    - + Process Communication
    - + Process Resource Usage
  - Stage B: Coordinated Multiprocessing Program
    - + Process Synchronization using Signals
    - + Process Termination
- Part Two [8 points]
  - Coordinated Multithreaded Program
    - + Threads Management
    - + Synchronization
    - + Producer/consumer interaction

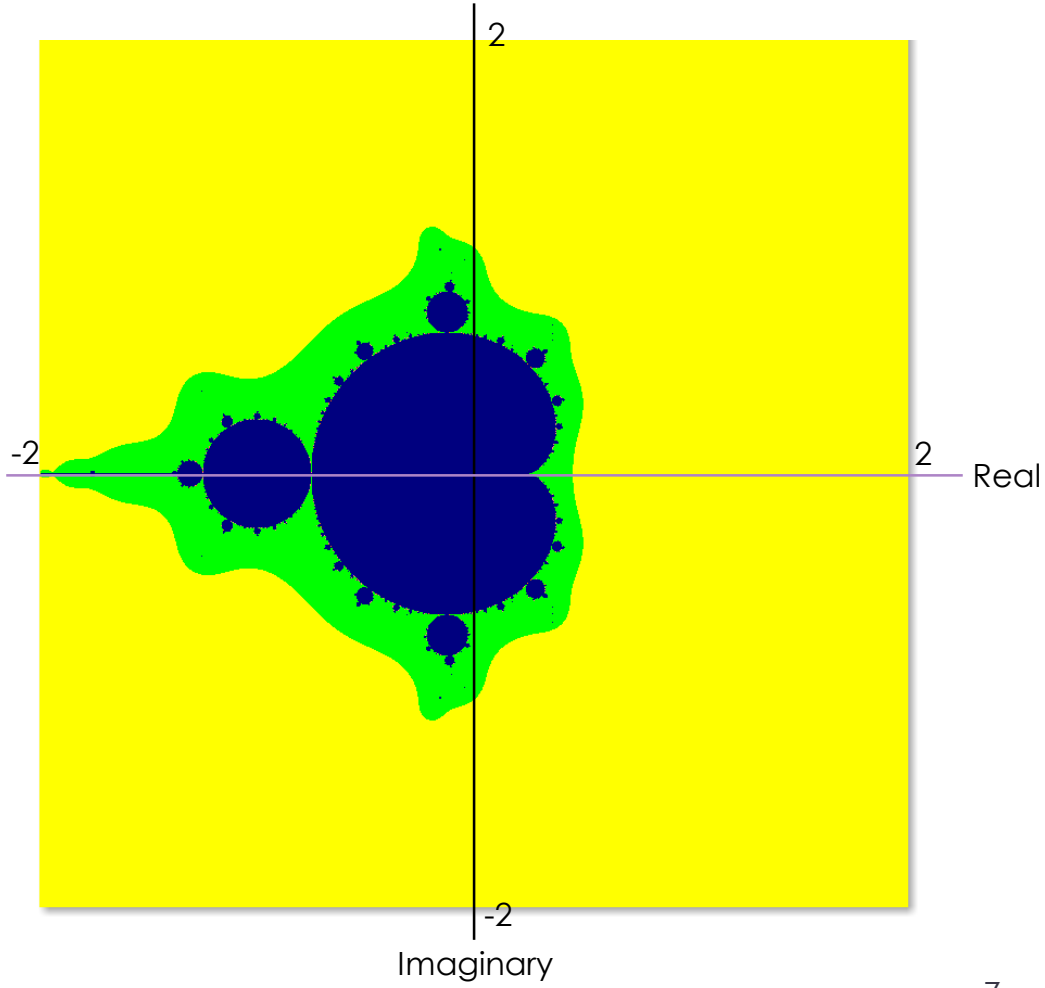
Mandelbrot Set Computation

- The Mandelbrot set is a famous example of a fractal in mathematics
- It is obtained from the iterative computation over the quadratic recurrence equation

$$z_{k+1} = {z_k}^2 + c$$

where $z_0 = 0$

- Mandelbrot set image is created by determining each sample point c in the complex plane goes to infinity or not



Sequential Program – part0-Mandelbrot.c

Measure the startup time of the program

Measure the starting time of the Mandelbrot computation

For each pixel in the 2D complex plane

    Determine the result value from the Mandelbrot computation

Measure the end time of the Mandelbrot computation and report the total computation time

Measure the end time and report the overall time

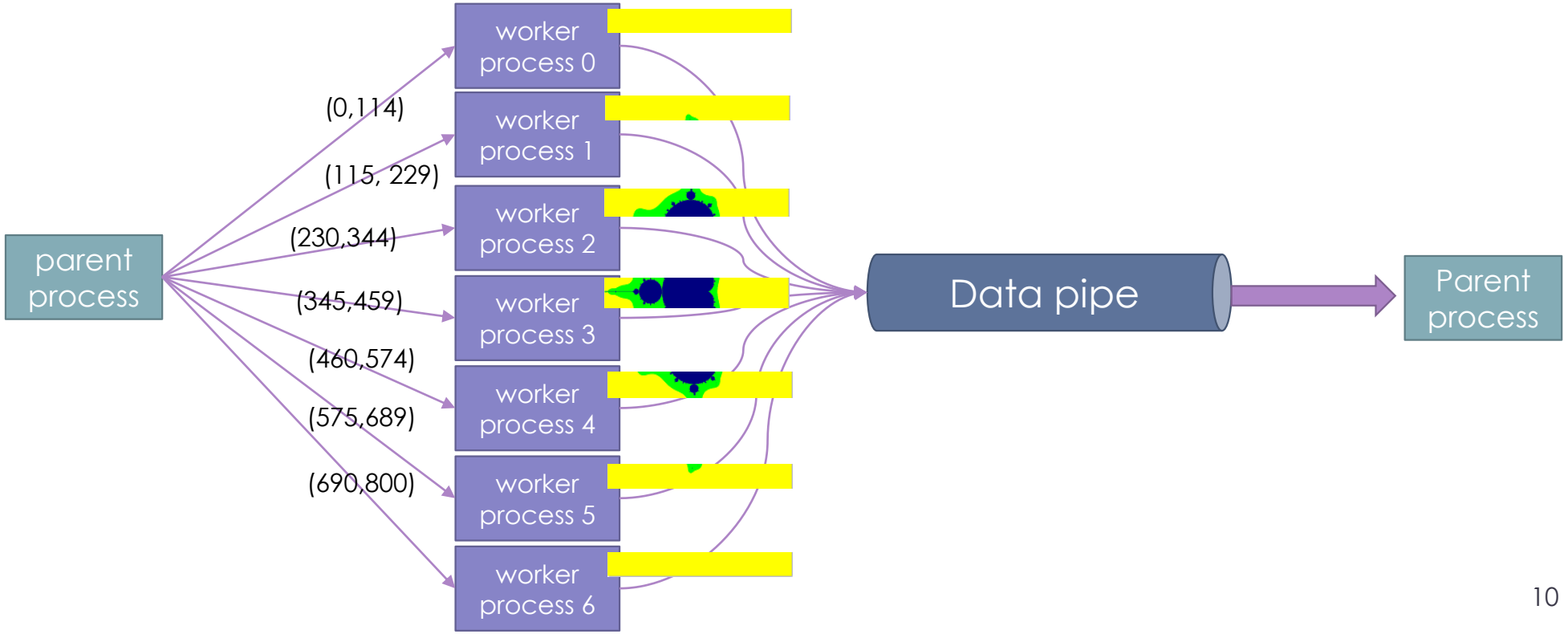
Draw the image

Part 0 Sample output

```
Start the computation ...
 ... completed. Elapse time = 24013.668 ms
Total elapse time measured by the process =
24013.723 ms
Draw the image
```

Part 1A – Simple Multiprocessing Program

- Parent process divides the computation workload into **blocks of rows** and assigns each block to a child process to work on
- Child processes return the computation results **row-by-row** back to the parent process through the Data pipe



Part 1A – Why returns row-by-row?

- All child processes make use of the Data pipe to return the computation results of the block of rows
- Will data from Child 1 mix with data from Child 2?
  - Yes, the contents would mix together if we place the whole block of data to the pipe as the pipe is shared by all child processes

- To avoid this issue, we take advantage of the **atomic guarantee** feature of the pipe:
  - Any data written to the pipe with size less that or equal to PIPE_BUF
  - For Linux, PIPE_BUF = 4096 bytes
  - Given a row has 800 pixels, each is a floating point number; that spans 3200 bytes

Structure of Part 1A Program

- Record program start time
  - Creation of pipe for collecting results
  - Create child processes and distribute tasks
  - Collect computation results
  - Wait for termination of child processes
  - Collect resource usage statistics
- Record program end time
- Print timing statistics
- Display image

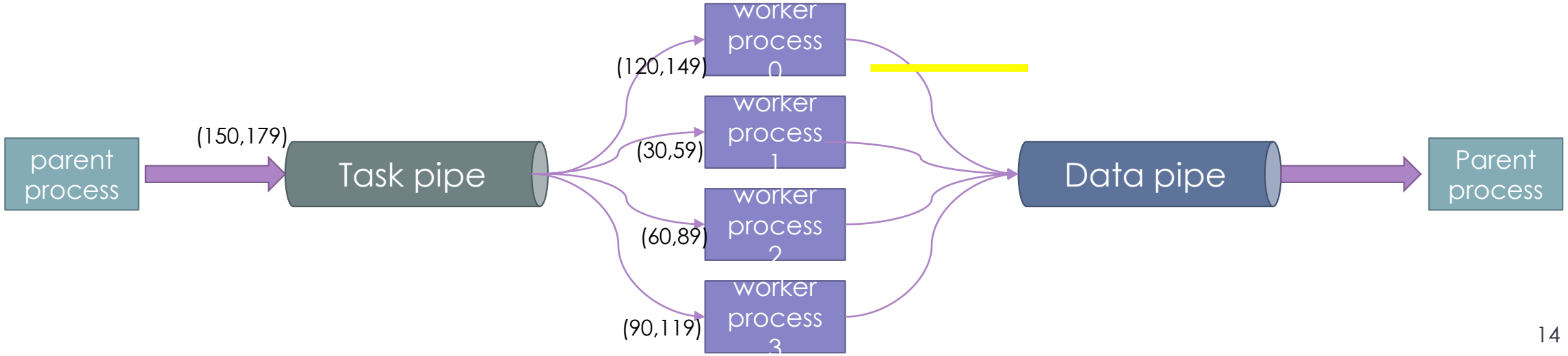
- For each child process
  - Record the start time
  - Perform the computation
  - Record the end time
  - Print the elapse time
  - Pass the results to the parent

Part 1A Sample output

```
Child(13993) : Start the computation ...
Start collecting the image lines
Child(13997) : Start the computation ...
Child(13995) : Start the computation ...
Child(13994) : Start the computation ...
Child(13996) : Start the computation ...
Child(13993) :       ... completed. Elapse time = 7.028 ms
Child(13997) :       ... completed. Elapse time = 7.432 ms
Child(13994) :       ... completed. Elapse time = 3507.499 ms
Child(13996) :       ... completed. Elapse time = 3629.017 ms
Child(13995) :       ... completed. Elapse time = 17327.031 ms
All Child processes have completed
Total time spent by all child processes in user mode = 24460.000 ms
Total time spent by all child processes in system mode = 4.000 ms
Total time spent by parent process in user mode = 4.000 ms
Total time spent by parent process in system mode = 8.000 ms
Total elapse time measured by parent process = 17330.760 ms
Draw the image
```

Part 1B – Coordinated Multiprocessing Program

- Parent process (boss) creates a fixed number of worker processes
- Boss distributes tasks via Task pipe and collect results via Data pipe
- Boss sends a signal to worker to inform it that a task is available in Task pipe
- Worker picks up a task and processes it after receiving the signal
- Boss informs workers to terminate by sending a termination signal



Structure of Part 1B Program

- Record program start time
  - Install signal handlers
  - Create Task pipe and Data pipe
  - Create child processes
  - Distribute tasks and signal child processes
  - Collect computation results and distribute tasks to idle child processes
  - Signal workers to terminate
  - Wait for termination of child processes and collect their exit statuses
  - Collect resource usage statistics
- Record program end time
- Print timing statistics
- Display image

- For each child process
  - Wait for SIGUSR1 signal
  - Get a task
  - Record the start time
  - Perform the computation
  - Record the end time
  - Print the elapse time
  - Pass the results to the parent
  - Repeat again

P1B Sample output

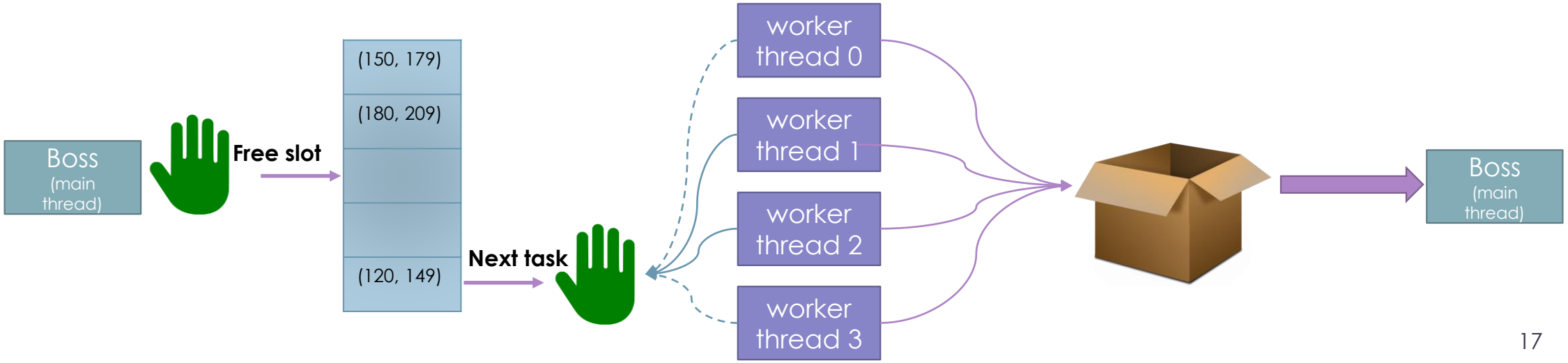
```
Child(21866) : Start up. Wait for task!
Child(21867) : Start up. Wait for task!
Child(21868) : Start up. Wait for task!
Child(21869) : Start up. Wait for task!
Start collecting the image lines
Child(21866) : Start the computation ...
Child(21868) : Start the computation ...
Child(21869) : Start the computation ...
Child(21867) : Start the computation ...
Child(21870) : Start up. Wait for task!
Child(21870) : Start the computation ...
Child(21866) :  ... completed. Elapse time = 0.876 ms
Child(21868) :  ... completed. Elapse time = 0.910 ms
Child(21867) :  ... completed. Elapse time = 0.954 ms
Child(21869) :  ... completed. Elapse time = 0.976 ms
Child(21870) :  ... completed. Elapse time = 1.006 ms
Child(21866) : Start the computation ...
Child(21867) : Start the computation ...
Child(21868) : Start the computation ...
Child(21869) : Start the computation ...
Child(21870) : Start the computation ...
Child(21866) :  ... completed. Elapse time = 1.026 ms
Child(21866) : Start the computation ...
Child(21867) :  ... completed. Elapse time = 1.044 ms
Child(21868) :  ... completed. Elapse time = 1.131 ms
Child(21869) :  ... completed. Elapse time = 1.148 ms
Child(21867) : Start the computation ...
Child(21868) : Start the computation ...
Child(21869) : Start the computation ...
Child(21866) :  ... completed. Elapse time = 5.031 ms
Child(21870) :  ... completed. Elapse time = 5.663 ms
Child(21866) : Start the computation ...
Child(21870) : Start the computation ...
Child(21867) :  ... completed. Elapse time = 141.016 ms
Child(21867) : Start the computation ...
Child(21868) :  ... completed. Elapse time = 389.282 ms
Child(21868) : Start the computation ...
```

```
Child(21869) :  ... completed. Elapse time = 499.403 ms
Child(21869) : Start the computation ...
Child(21866) :  ... completed. Elapse time = 1361.569 ms
Child(21866) : Start the computation ...
Child(21870) :  ... completed. Elapse time = 1568.870 ms
Child(21870) : Start the computation ...
Child(21867) :  ... completed. Elapse time = 1782.524 ms
Child(21867) : Start the computation ...
Child(21868) :  ... completed. Elapse time = 2054.256 ms
Child(21868) : Start the computation ...
Child(21869) :  ... completed. Elapse time = 2583.502 ms
Child(21869) : Start the computation ...
Child(21866) :  ... completed. Elapse time = 2798.525 ms
Child(21866) : Start the computation ...
Child(21870) :  ... completed. Elapse time = 2719.551 ms
Child(21870) : Start the computation ...
Child(21867) :  ... completed. Elapse time = 2559.109 ms
Child(21867) : Start the computation ...
Child(21868) :  ... completed. Elapse time = 2131.709 ms
Child(21868) : Start the computation ...
Child(21869) :  ... completed. Elapse time = 1805.886 ms
Child(21869) : Start the computation ...
Child(21868) :  ... completed. Elapse time = 338.953 ms
Child(21868) : Start the computation ...
Child(21868) :  ... completed. Elapse time = 1.636 ms
Child(21868) : Start the computation ...
Child(21868) :  ... completed. Elapse time = 3.121 ms
Child(21868) : Start the computation ...
Child(21868) :  ... completed. Elapse time = 0.636 ms
Child(21868) : Start the computation ...
Child(21868) :  ... completed. Elapse time = 0.628 ms
Child(21868) : Start the computation ...
Child(21868) :  ... completed. Elapse time = 0.553 ms
Child(21868) : Start the computation ...
Child(21868) :  ... completed. Elapse time = 0.540 ms
Child(21868) : Start the computation ...
Child(21868) :  ... completed. Elapse time = 0.536 ms
```

```
Child(21868) : Start the computation ...
Child(21868) :  ... completed. Elapse time = 0.535 ms
Child(21868) : Start the computation ...
Child(21868) :  ... completed. Elapse time = 0.534 ms
Child(21868) : Start the computation ...
Child(21868) :  ... completed. Elapse time = 0.495 ms
Child(21868) : Start the computation ...
Child(21868) :  ... completed. Elapse time = 0.479 ms
Child(21867) :  ... completed. Elapse time = 502.480 ms
Child(21869) :  ... completed. Elapse time = 113.751 ms
Child(21870) :  ... completed. Elapse time = 1414.422 ms
Child(21866) :  ... completed. Elapse time = 1602.743 ms
Process 21866 is interrupted by ^C. Bye Bye
Process 21867 is interrupted by ^C. Bye Bye
Process 21868 is interrupted by ^C. Bye Bye
Process 21869 is interrupted by ^C. Bye Bye
Process 21870 is interrupted by ^C. Bye Bye
Child process 21866 terminated and completed 6 tasks
Child process 21868 terminated and completed 17 tasks
Child process 21869 terminated and completed 6 tasks
Child process 21870 terminated and completed 5 tasks
Child process 21867 terminated and completed 6 tasks
All Child processes have completed
Total time spent by all child processes in user mode =
26400.000 ms
Total time spent by all child processes in system mode =
28.000 ms
Total time spent by parent process in user mode = 0.000
ms
Total time spent by parent process in system mode =
12.000 ms
Total elapse time measured by parent process = 5777.566
ms
Draw the image
```

Part 2 – Coordinated Multithreaded Program

- Main process (boss) creates a fixed number of worker threads
- Boss distributes tasks via a bounded buffer and workers pick up tasks from it
  - They are coordinating by using the Producer/Consumer interaction
- Implement a mechanism for workers to return counting results to the boss
- Implement a mechanism for workers to detect termination



Structure of Part 2 Program

- Record program start time
  - Create worker threads
  - Distribute tasks via bounded-buffer
  - Inform workers to terminate
  - Wait for termination of worker threads and collect their exit statuses
  - Collect resource usage statistics
- Record program end time
- Print timing statistics
- Display image

- For each worker thread
  - Get a task for the bounded-buffer
  - Record the start time
  - Perform the computation
  - Record the end time
  - Print the elapse time
  - return the results
  - Terminate or Repeat again

P2 Sample output

```
Worker(0) : Start up. Wait for task!
Worker(1) : Start up. Wait for task!
Worker(1) : Start the computation ...
Worker(1) :       ... completed. Elapse time = 0.669 ms
Worker(1) : Start the computation ...
Worker(1) :       ... completed. Elapse time = 0.928 ms
Worker(1) : Start the computation ...
Worker(1) :       ... completed. Elapse time = 0.714 ms
Worker(1) : Start the computation ...
Worker(1) :       ... completed. Elapse time = 0.733 ms
Worker(1) : Start the computation ...
Worker(3) : Start up. Wait for task!
Worker(4) : Start up. Wait for task!
Worker(2) : Start up. Wait for task!
Worker(0) : Start the computation ...
Worker(3) : Start the computation ...
Worker(2) : Start the computation ...
Worker(4) : Start the computation ...
Worker(1) :       ... completed. Elapse time = 0.760 ms
Worker(1) : Start the computation ...
Worker(0) :       ... completed. Elapse time = 0.690 ms
Worker(0) : Start the computation ...
Worker(3) :       ... completed. Elapse time = 0.832 ms
Worker(3) : Start the computation ...
Worker(4) :       ... completed. Elapse time = 0.927 ms
Worker(4) : Start the computation ...
Worker(2) :       ... completed. Elapse time = 1.035 ms
Worker(2) : Start the computation ...
Worker(0) :       ... completed. Elapse time = 5.320 ms
Worker(0) : Start the computation ...
Worker(1) :       ... completed. Elapse time = 5.851 ms
Worker(1) : Start the computation ...
Worker(3) :       ... completed. Elapse time = 139.569 ms
Worker(3) : Start the computation ...
Worker(4) :       ... completed. Elapse time = 379.427 ms
```

```
Worker(4) : Start the computation ...
Worker(2) :       ... completed. Elapse time = 527.476 ms
Worker(2) : Start the computation ...
Worker(0) :       ... completed. Elapse time = 1340.056 ms
Worker(0) : Start the computation ...
Worker(1) :       ... completed. Elapse time = 1586.350 ms
Worker(1) : Start the computation ...
Worker(3) :       ... completed. Elapse time = 1712.583 ms
Worker(3) : Start the computation ...
Worker(4) :       ... completed. Elapse time = 1991.579 ms
Worker(4) : Start the computation ...
Worker(2) :       ... completed. Elapse time = 2507.898 ms
Worker(2) : Start the computation ...
Worker(0) :       ... completed. Elapse time = 2742.174 ms
Worker(0) : Start the computation ...
Worker(3) :       ... completed. Elapse time = 2447.473 ms
Worker(3) : Start the computation ...
Worker(4) :       ... completed. Elapse time = 2017.753 ms
Worker(4) : Start the computation ...
Worker(1) :       ... completed. Elapse time = 2802.177 ms
Worker(1) : Start the computation ...
Worker(1) :       ... completed. Elapse time = 292.608 ms
Worker(1) : Start the computation ...
Worker(2) :       ... completed. Elapse time = 1723.355 ms
Worker(2) : Start the computation ...
Worker(2) :       ... completed. Elapse time = 1.341 ms
Worker(2) : Start the computation ...
Worker(2) :       ... completed. Elapse time = 2.464 ms
Worker(2) : Start the computation ...
Worker(2) :       ... completed. Elapse time = 0.454 ms
Worker(2) : Start the computation ...
Worker(2) :       ... completed. Elapse time = 0.411 ms
Worker(2) : Start the computation ...
Worker(2) :       ... completed. Elapse time = 0.373 ms
Worker(2) : Start the computation ...
```

```
Worker(2) :       ... completed. Elapse time = 0.364 ms
Worker(2) : Start the computation ...
Worker(2) :       ... completed. Elapse time = 0.357 ms
Worker(2) : Start the computation ...
Worker(2) :       ... completed. Elapse time = 0.339 ms
Worker(2) : Start the computation ...
Worker(2) :       ... completed. Elapse time = 0.289 ms
Worker(2) : Start the computation ...
Worker(2) :       ... completed. Elapse time = 0.277 ms
Worker(2) : Start the computation ...
Worker(2) :       ... completed. Elapse time = 0.257 ms
Worker(1) :       ... completed. Elapse time = 109.225 ms
Worker(4) :       ... completed. Elapse time = 436.735 ms
Worker(3) :       ... completed. Elapse time = 1249.937 ms
Worker(0) :       ... completed. Elapse time = 1542.625 ms
Worker thread 0 has terminated and completed 5 tasks.
Worker thread 1 has terminated and completed 10 tasks.
Worker thread 2 has terminated and completed 15 tasks.
Worker thread 3 has terminated and completed 5 tasks.
Worker thread 4 has terminated and completed 5 tasks.
All worker threads have terminated
Total time spent by process and its threads in user mode =
25560.000 ms
Total time spent by process and its threads in system
mode = 4.000 ms
Total elapse time measured by the process = 5636.160 ms
Draw the image
```

Submission Deadlines

	Deadline	Submission
Part One	Nov 1, 2017 (Wed) 5:00 pm	• part1a-mandelbrot.c (**optional but recommended**) • part1b-mandelbrot.c
Part Two	Nov 22, 2017 (Wed) 5:00 pm	• part2-mandelbrot.c

Late submission policy: At most 3 days with 10% penalty for each day of delay.

Submission of Assignment

- You should name your program "partxx-mandelbrot.c" for each part of the project
- Submit your programs to the course's Moodle web site under the corresponding submission pages
- Add your signature in the header of each submitted program and make clear documentation

```
/*************************************************************
* Filename: part1b-mandelbrot.c
* Student name: Harry Potter
* Student no.: xxxxxxxxxx
* Date: Oct 26, 2017
* version: 1.1
* Development platform: Course VM
* Compilation: gcc part1b-Mandelbrot.c -l SDL2 -lm
*************************************************************/
```

Grading Criteria

	Criteria
Part 1 (9 points)	• Documentation • Program specific requirements • processes are running in parallel; child process displays the time spent in handling a task; use signals to coordinate and control processes; display the termination statuses of workers; parent waits for termination of all child processes before outputs results • Display total elapsed time • No processes left behind after program termination
Part 2 (8 points)	• Documentation • Program specific requirements • fixed number of worker threads; use producer/consumer interaction; worker displays the time spent in handling a task; collect workers' termination status, main process waits for termination of all threads before outputs results • Display total elapsed time