CSC3002

Introduction to Computer Science: Programming Paradigms

Project Proposal

Jie Liu 117010169

Siheng Wang 117010264

Gaoyuan Xu 117010331

Sijing Yu 117010363

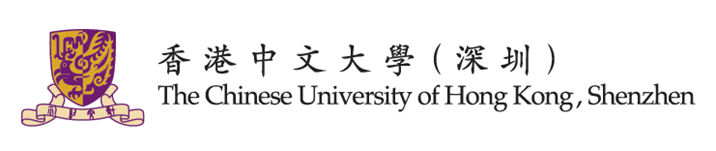
Jin Zhang 117010377

Lingxi Ji 118010112

Course Coordinator: Prof. Rui Huang

The Chinese University of Hong Kong, Shenzhen

Spring 2020



1. **Introduction**

Currently, there are many operating systems like Windows, MacOS and Linux. We can store files and run programs in these systems. However, operating system itself is a large program which is not editable in the user mode. The purpose of this project is to simulate an operating system to learn how it is developed.

The project is designed with four components and written in C language, which can be adapted into other languages. The first module is memory management, which execute the command of allocate and deallocate memory for program execution. The second module is task scheduling, which concerns the arrangement of various tasks in the system. The third module is file system, which allows users to store and open files. The last module is exceptions and error handling. When the system is running, it is inevitable to raise error and needs the system to fix problem. The project is presented in a GUI/TUI environment, which allows us to run processes and test features.

1. **Project Background**

CSC3002 is an introductory course taken by students in CUHKSZ who have had 1-year programming course. This course includes the several core concepts, such as abstraction, representation, algorithmic efficiency and program paradigms. As the main outcome of this course, our group thinks it should reflect how we learn about these concepts. Thus, we choose to implement a simulation of operation system (OS) with the following two reasons:

* C/C++ language has a lower level than python language in programming, it helps us learn more about how basic applications and software works and how they interact with computer.
* The completion of this projects requires not only the implementation of a well-designed OS but also a good understanding of how we improve the efficiency of program and optimize the data structure, though it is complicated.

1. **Resource and Implementation**
   1. **UI Design**

The main user interface is the desktop graphical user interface, which mimics the desktop interface of a prevailing operating system, such as Windows. In the desktop interface, there are four main components lining up along the left, namely **“My Documents”**, **“My Memory”**, **“Task Scheduler”**, **“Terminal”**.

My Documents is a folder which simulates a file folder of the real operating system. The main usage or purpose as we design is to contain all the files created by the user. Some functionalities we are going to support include creating, editing, deleting, etc.

Simply put, My Memory is just a button. After the user clicks My Memory, an image is going to image on the desktop. Some fundamental information will be presented to the user in the form of pie chart with corresponding captions of absolute values and proportions, such as the total amount of the memory and each ongoing task.

Task Scheduler is the component that provides the ability to schedule the launch of programs or scripts at pre-defined time or after specified time.

Terminal is well similar to the terminal component of Linux or macOS. Terminal on the desktop is simply a button as well. After being clicked by the user, the terminal interface will appear. Terminal support a bundle of functionalities, such as creating a file, opening a file, deleting a file, checking the memory usage, ending a task, etc. Each input of Terminal should be a well-defined command.

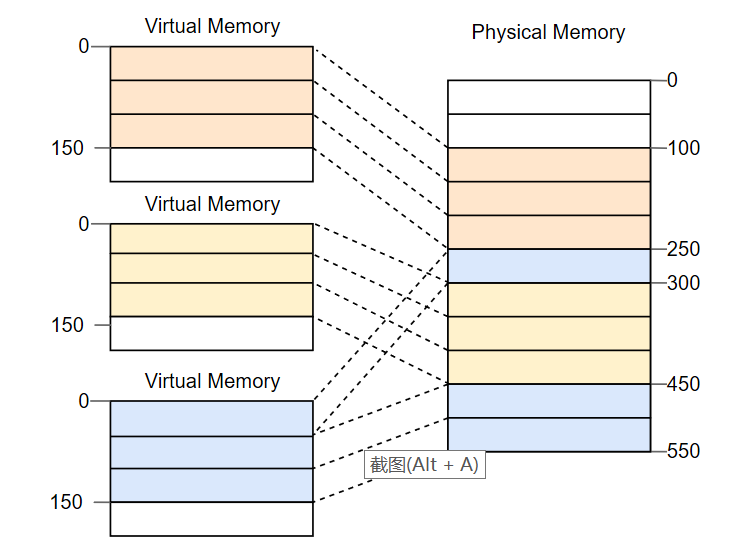
Taking the consideration of designing a desktop graphical user interface, we are going to implement Qt Designer as the main assistance to visualize and realize the design of the graphical user interface. As widely suggested, we are going to study *C++ GUI Programming with Qt 4, Second Edition*, to learn how to use Qt write GUI programs. Referred to the mainstream design of GUI applications, some advanced tools and packages such as XML, OpenGL and QImage to polish the graphical user interface are with the scope of our consideration.

* 1. **Memory Allocation**

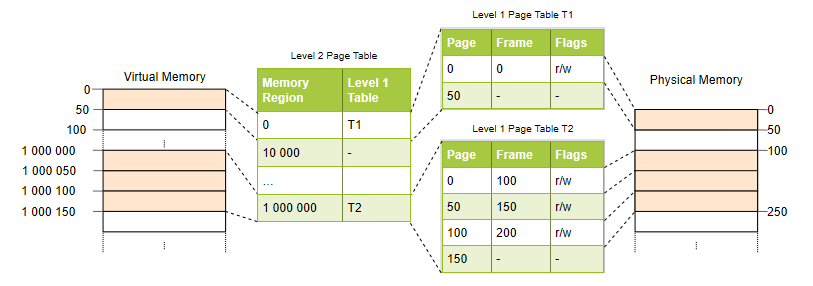
The memory allocator in OS takes the responsibility of properly distribute, release and protect the memory zoom. Nowadays, there are several frequently used memory allocation strategies, containing relocation, segmentation, paging and virtual memory. There are plenty of memory allocation structures deduced by those strategies, all of them have their pros and cons. In this project, we mainly focus on the **multi-level page tables** as well as **virtual memory** to implement our allocator. The advantages of such model are stated below.

One main task of our allocator is to isolate programs from each other, which means we need to ensure that the memory area of one process will never be accessible for other process while it is still working. To achieve this goal, we need a memory protection technique. Two powerful techniques are segmentation and paging. While segmentation uses variable-sized memory regions and suffers from external fragmentation, the paging uses fixed-sized pages and allows much more fine-grained control over access permissions. Though paging strategy still have the problem of inner fragmentation, it is not compatible with the external fragmentation caused by segmentation, as well as the memory relocation problems. While a linear one-level page table suffers the problem of its large size, the multi-level page tables can minimize the memory occupied by the table itself by dividing and indexing.

The other aspect is the usage of virtual memory, Virtual memory is a technique that allows the execution of processes that are not completely in memory. One major advantage of this scheme is that programs can be larger than physical memory. Since the execution of a program needs continuous memory, we can allocate them with continuous virtual memory then map the virtual memory to the physical memory where the mapping can be discrete, in this way we make full use of memory space, as shown in the diagram below.



The mechanism of our model can be interpreted by the diagram below:

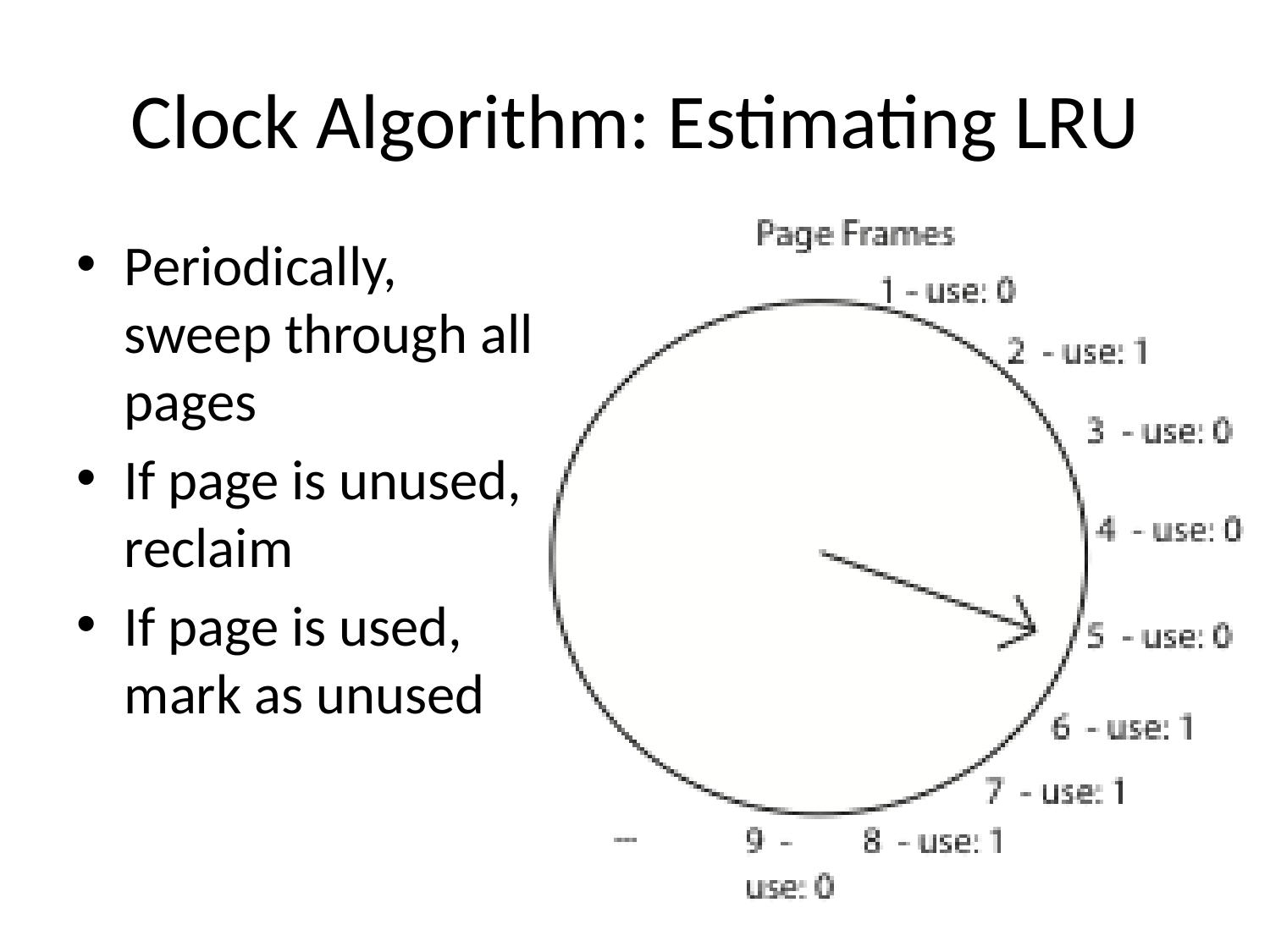


The core idea of the model is we allocate virtual memory to programs then use the multi-level page table iteratively to access its mapping physical memory. When we access a page but cannot find its corresponding physical memory (page fault), we swap some pages in physical memory with the pages we need in hard disk based on our paging swapping algorithm.

The implementation of this memory allocation model can be mainly divided into three parts:

1. The implementation of virtual memory
2. The MMU (Memory Management Unit) system, which has the function of translating the virtual address to physical address.
3. Paging system with swapping technique

We will use some powerful data structures like double linked list and heap. The final outcome depends on their performance. Moreover, an essential part in the implementation is about the paging swapping algorithm. We now decide to use Clock page replacement algorithm to implement this function, the implementation diagram of this algorithm shows below.



There are other swapping algorithms like Optimal Page Replacement Algorithm, Not Recently Used Replacement Algorithm, First-In,First-Out Page Replacement Algorithm, Second Chance Page Replacement Algorithm and LRU Page Replacement Algorithm. Any further adjustment could be done according to the performance of our algorithm.

* 1. **Task Scheduling**

Generally, high-level scheduling is job scheduling, and low-level scheduling is process scheduling. The function of task scheduling is to determine which process in the ready queue gets how many computing resources. There are mainly two modes for task scheduling: non-preemptive-mode (forbid suspending) and preemptive-mode (allow suspending)

Multiple algorithms can be applied to task scheduling.

1. First come first serve (FCFS)

Select a process from the ready queue that enters the queue first, assign a processor to it, and put it into operation. The process does not abandon the processor until it has completed, or an event has blocked it.

2. Shortest Job First (SJF)

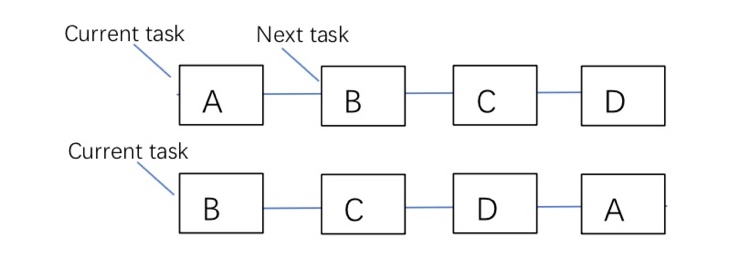
Select a job/process with the shortest estimated running time from the reserve queue/ready queue and assign the processor to it so that it can execute immediately and continuously until it is completed, or reschedule when an event occurs, and the processor is blocked and abandoned.

3.Priority Scheduling

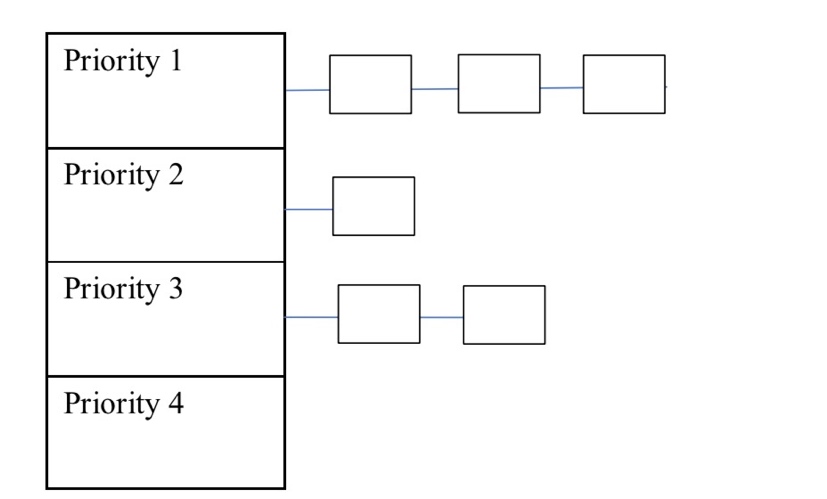
The system will select several jobs with the highest priority from the backup queue to load into memory. When it is used for process scheduling, the algorithm assigns the processor to the process with the highest priority in the ready queue.4. Round Robin

4. Round Robin

The system arranges all the ready processes into a queue according to the principle of first come first serve. Each time when scheduling, the CPU is assigned to the first process of the queue, and it is ordered to execute a time slice. When the execution time slice is used up, a timer sends out a clock interrupt request, and the scheduler stops the execution and sends it to the end of the ready queue. It ensures that the system can respond to all users' requests within a given time.



We decide to use priority scheduling algorithm (dynamic priority mode). If several tasks have same priority, then apply round robin algorithm.



Appropriate data structure like priority queue, linked list and heap, the final decision will be based on their performance after tests.

The module is expected to:

1. create tasks

2. assign arrival time and burst time, computing waiting time and turnaround time.

3. sorted the processes based on burst time

4. finding the next process according to arrival time

5. call tasks

* 1. **File System**

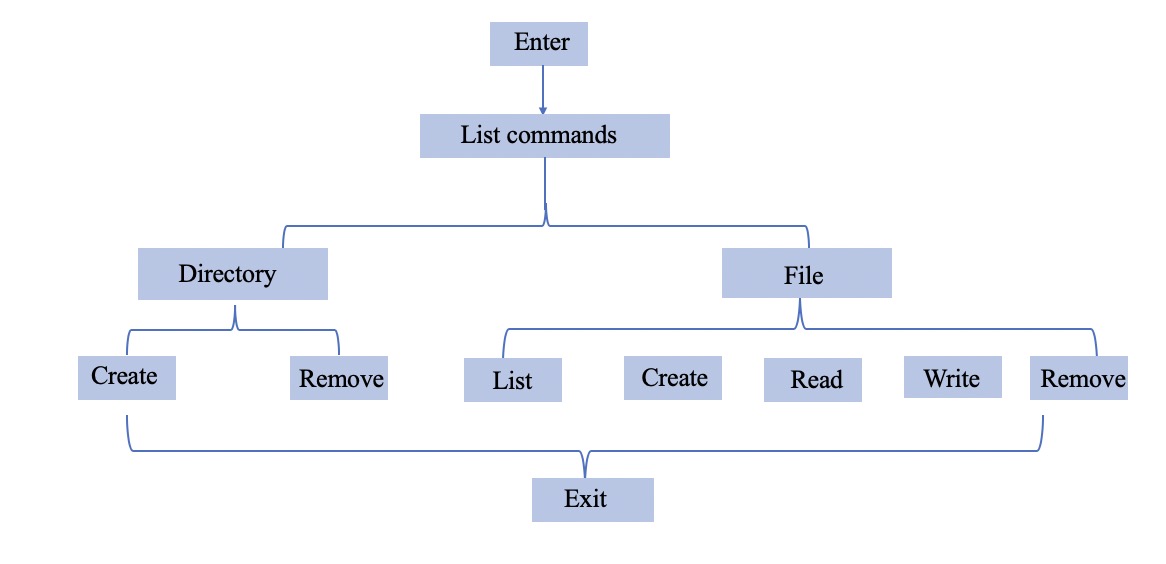
File system is the core component of operation system, which manages and stores file information. In this project, our group will program a file system, and simulate the human-computer interaction process of file management.

The implementation of file system includes managing directory, listing files, creating files, reading files, writing files, deleting files and other functions. We will set a limited size of volume in the beginning, then define the maximum number of file and the file’s maximum size. Users type given commands in the terminal so that the file system can find the files’ location and perform operation on them.

The detailed commands are shown below:

|  |  |
| --- | --- |
| **Command** | **Description** |
| help | Show all the commands in file system |
| list | List all the files’ name in the current folder |
| vim | Create a new file in the current folder |
| open\_r | Open a read-only file |
| open\_w | Open the file and write content in the file |
| remove | Delete the file and release the space |
| mkdir | Create new directory and file folder |
| cd .. | Move to the root directory or the absolute directory |

The basic function chart of the file system is shown below:



* 1. **Exception System**

Error handling can be divided into two parts: response and recovery procedures. It anticipates, detects and solves different types of errors.

Although there could be various errors, we will only consider a subset of them due to time limit and our simulated OS is so simple that it might not generate many errors. Specifically, errors could occur, but not limted in, when:

- File System: open/write/delete file without permission; quit without saving; attempt to use duplicated file names.

- Task Scheduling & Memory allocation: memory usage exceeds limit; process breaks down; memory allocation overlapping.

- Terminal: use of invalid instructions; file path not exists.

……

Different errors should trigger different responses. If the error is fatal, such as memory usage exceeding limit, our OS will cease all processes and start again. Otherwise, if the error is minor, our OS simply throws out a warning message.

1. **Time Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Week** | **Date** | **Activity** | **Content** |
| 9 | 3/23-3/29 | Project proposal due | Finish the proposal |
| 10 | 3/30-4/05 | Assignment 2 | Start writing codes to implement all OS components |
| 11 | 4/06-4/12 | Assignment 2 due |
| 12 | 4/13-4/19 | Assignment 3 |
| 13 | 4/20-4/26 | Assignment 3 due | integrate codes |
| 14 | 4/27-5/03 | Assignment 4 | Start writing report |
| 15 | 5/04-5/10 | Assignment 4 due | Test codes and adjustment |
| 16 | 5/11-5/17 | Project and report due | Finish the report |

1. **Work Distribution**

|  |  |  |
| --- | --- | --- |
| Name | Student Number | Division |
| Jie Liu | 117010169 | Memory Allocation |
| Siheng Wang | 117010264 | GUI Design |
| Gaoyuan Xu | 117010331 | Document Writing and Codes Synchronization |
| Sijing Yu | 117010363 | File System |
| Jin Zhang | 117010377 | Exceptions, Error Handling |
| Lingxi Ji | 118010112 | Task Scheduling, Thread Scheduling |

1. **References**

**Websites:**

1. https://os.phil-opp.com/paging-introduction/
2. https://blog.csdn.net/qq\_32635069/article/details/74838187
3. https://en.cppreference.com/w/cpp/header/filesystem

**Figures cited from**

1. https://os.phil-opp.com/paging-introduction/ Introduction to paging
2. https://slide-finder.com/view/Caching-and-Virtual-Memory.254456.html