

COM 326: Operating Systems

Group 1: Winchester 11

Project Manager - **Jaleel Watler**

System Architect - **Khanh Nghiem**

Lead Software Designer - **Nate Devine**

Lead Software Engineer - **Elijah Pineda**

Business Analyst - **Johnathan Evanilla**

Quality Tester - **Sam Barnes**

Software Design Document

Introduction

This document aims to provide the technical documentation and design justification for the operating system simulator built by the Winchester 11 team - the Winch11 OS-Sim 1.0.

Readers should find information about system overview, requirement, design considerations, architectural strategies, system architecture, subsystem architecture, policies and tactics, detailed system design, detailed subsystem design, test plan, glossary and bibliography.

For any information not specified above, please contact the project manager.

This document is suitable for client consultation, leadership report, and future engineer onboarding.

Requirements

1. Phase 1 - First prototype of the Process Control Block (PCB)

For phase 1, the team is required to build

...a well-designed and -documented code base;

...an appropriate data structure for each computer process. Each process must at least contain these data:

- Unique process ID (pid)
- Priority
- Current state (active/inactive)
- Time created
- Mode (user/kernel);

...a prototype for the PCB that has at least the capabilities to:

- Read process descriptions for file
- Store processes in a queue
- Print out a list of active processes
- Update existing processes and add new processes;

...a user interface that is:

- Light and intuitive

In our implementation, we assume that the user is familiar with a command line interface.

System Overview

The Winch11 OS-Sim 1.0 is built with Python 3.6 and can be executed through any Python shell or UNIX terminal.

Design Considerations

1. Assumptions and Dependencies:

- User has experience using a command line interface
- Python 3.6 or later, Python shell or terminal are installed on the user's machine which runs the simulator
- The machine has available memory and storage capacity

2. General Constraints:

- Compliance to coding conventions and documentation expectation
- Attention to private and public attributes and methods
- Rigorous unit testing and debugging
- Centralized version control and continuous integration best-practices
- Appropriate API design to maximize usability and protect data security
- Scalable, efficient algorithms that are robust for large data input size
- Available mechanism to user interruption

3. Goals and Guidelines:

- Emulate contemporary popular operating systems
- Conduct close research into current best practices and real world design considerations before implementation
- Provide a light, intuitive user interface
- Keep the code base light and easy to maintain
- Prefer third-party libraries over self-implementation

4. Development Methods:

- The team employs an agile development framework, continuously produces code, tests, debugs, and adapts to new requirements from the client. The code base is built using the object oriented design approach for code reusability and modularity.

Architectural Strategies

1. General Strategies:

- Agility over efficiency: Prefer high level programming languages and frameworks at the expense of implementation efficiency. Due to this trade-off decision, the development team can put more focus on design/algorithms rather than syntax, and respond more quickly to new requirements, bugs, or errors
 - 2. Use of products:
 - Programming language: Python 3.x (3.6.5 distributed by Anaconda)
 - Version control: git 2.x (2.15.1)
 - Cloud-based distributed version control: GitHub
 - 3. User interface:
 - Early phases: command line interface via Python
 - Later phases: graphic user interface (*planned*)
 - Graphics libraries for consideration: Tkinter, wxPython, Kivy, PyQt
 - 4. Version control, error detection and recovery:
 - Separate development and production branches in code repository
 - Only the lead software developer has privilege to push code base from development to production
 - Each developer maintains their own branch and must send a pull request to the lead developer. Code can only be pushed to production after all merge conflicts have been resolved and all test cases are passed
 - Revert to the last stable version in production when serious errors are reported
-

System Architecture

The system is composed of a variety of files and data structures. The subsystem architecture is described below.

Files

Control.py: This file is responsible for creating an interface through which the user can interact with the program.

PCB.py: This file contains the Process and PCB class methods.

PCB_utils.py: The PCB_utils.py function holds many of the functions used in the PCB. It is exclusively used by the PCB, and organizationally can be considered as part of the PCB class.

Data Structures

Process: The process class is responsible for initializing a process with the necessary attributes: ID, Activity, Priority, Birthday and Mode. The ID attribute is an integer value that serves as the

unique identifier for each process. The Activity attribute is a boolean variable that indicates whether or not the process is active. The Priority attribute is an integer value that specifies the priority level of the process. The Birthday attribute is simply the time at which the process was created. Finally, the Mode attribute is a boolean variable that indicates whether or not the process can be run in User mode. There are a variety of methods within the Process class that are able to get and update attributes of a specified process. All attributes for a given process, besides the unique id, are mutable.

Process Control Board: The PCB class is responsible for the input and output of processes, as well as handling any changes to the process queue, and the validation of all processes. Any user input and processing of that user input is handled by the PCB. Below are the relevant functions of the PCB

File Reading: Read in processes from an input .txt file. Any incorrectly formatted processes will not be accepted as a valid process. This will also sort the processes based on priority, and add them to a queue.

Add/Update: The PCB can handle user input to add a new process, or update an existing process. Added processes are verified to make sure valid parameters are given. Updating a process involves changing one of the process parameters.

Output: Active processes as well as process information can be output. The process information is the key, priority, time of creation, mode, and whether or not it is active.

Policies and Tactics:

- Choice of which specific product to use (compiler, interpreter, database, library, etc. ...) :
 - Queue library
- Engineering trade-offs:
 - We prioritized the functionality and simplicity of our code so at this point in time we haven't made any situational decisions that involve diminishing or losing one quality, quantity or property of a set or design in return for gains in other aspects.
- Plans for ensuring requirements traceability:
 - We kept the code simple, everything is modular (broken into classes) making it easy to follow which in turn makes it easy to check for requirements.
- Plans for testing the software:

- For testing purposes, we created lists of processes to have our PCB read in. One, containing processes with all of the required attributes, passes with no errors. Others, that are missing various attributes such as status, or id, will throw errors and not be accepted as processes.
- Plans for maintaining the software:
 - Generalize the code
 - Error Checking

Detailed System Design

1. Classes Documentation

1.1. Process Class

Description: each object represents a computer process. Each process has a unique identifier `id` called process ID and denoted as `pid`. This identifier is used in managing the life cycle of a process, including creating a new process or aborting a running process. Other data points include activity, relative priority, start time, and control mode (kernel/user)

Properties/Instance variables:

key	type <code>int</code> , range 00000-99999, unique identifier
active	type <code>boolean</code> , values <code>True-False</code>
priority	type <code>int</code> > 0, lower values indicate higher priority
time_created	type <code>string</code> , read-only access, records the time stamp when the process is forked
mode	type <code>boolean</code> (T = user, F = kernel)

Methods:

getKey	return <code>pid</code>
getPriority	return priority
getTimeCreated	return time created
getMode	return mode
isActive	return activity
setPriority	change the current priority
setMode	change the mode
setActive	change the activity

setTimeCreated change time created

1.2. **Priority Queue Class**

Description: imported from Python standard utility library, stores processes as nodes, each with a priority, provides access to the process with highest priority

1.3. **Process Control Block (PCB) Class**

Description: each object represents a computer process. Each process has a unique identifier `id` called process ID and denoted as `pid`. This identifier is used in managing the life cycle of a process, including creating a new process or aborting a running process. Other data points include activity, relative priority, start time,

Properties/Instance variables:

processes	type <code>list</code> , a ordered list of processes read from file
PCBqueue	type <code>queue</code> , a first-in-first-out list in order of priority

Methods:

printQueue	Prints all processes (active and inactive) in the queue
print_active_processes	Prints all active processes in the queue
print_process_info	Prints each attribute of a desired process
searchProcesses	Searches for a process by ID, returns false if it does not exist
readFile	Reads in processes from a file
Verify	Makes sure a process is in the process list before operating on it
add	Accepts new processes as input through user interface
updateProcessInfo	Updates attributes of the desired process

Test Plan

For testing purposes, we created lists of processes to have our PCB read in. One, containing processes with all of the required attributes, passes with no errors. Others, that are missing various attributes such as status, or id, will throw errors and not be accepted as processes.

```
winch11 — Python PCB.py — 95x32
Nates-MacBook-Pro:winch11 ndevine$ python3 PCB.py
Please enter the filename of the file you would like to read in: faketestfile
Invalid file name.
Please enter the filename of the file you would like to read in: faketestfile.txt

--- READING FILE ---

File not found. Try again
Please enter the filename of the file you would like to read in: dataset.txt

--- READING FILE ---

Not acceptable hours
Process ID 3546 isn't valid. Moving on to the next process...
Process ID 2446 has been validated
Process ID 2546 has been validated
Process ID 5154 isn't valid. Moving on to the next process...
Process ID 3135 isn't valid. Moving on to the next process...
Process ID  isn't valid. Moving on to the next process...
Process ID  isn't valid. Moving on to the next process...

--- FINISHING READING FILE ---

You can add a new process, update a process, see active processes or finish with the program
Please enter "update", "add", "print". Or enter "done" if you're finished with the program: █
```

Above, this screenshot shows the ability of our code to add new processes from a file. As can be seen in the screenshot, only a file that actually exists can be inputted for selection. Validity checks are performed on each parameter of the process definition, and a process is only validated and accepted if every parameter passes these checks.

```
winch11 — Python PCB.py — 79x49
Please enter "update", "add", "print". Or enter "done" if you're finished with the program: add

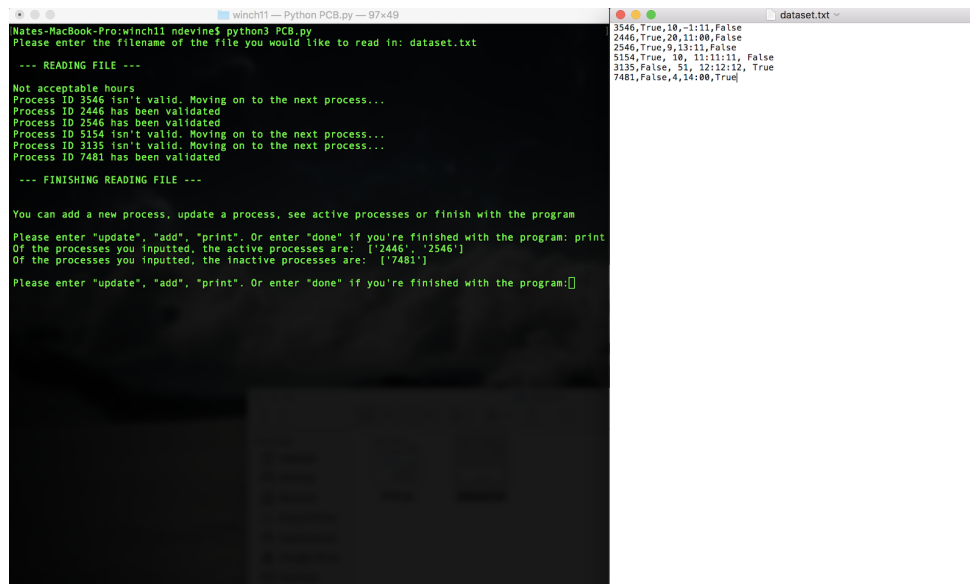
Current process queue:
Process ID: 2446, Priority: 20
Process ID: 2546, Priority: 9
Process ID: 7481, Priority: 4

Create a process, "True" or "False": True
Please enter a process ID: 2446
Error: ID 2446 already exists
Please enter a process ID: invalid input
Not an integer
Please enter a process ID: 9999
Please enter if process is active, "True" or "False": invalid input
Not an acceptable boolean
Please enter if process is active: True
Please enter process' priority, an integer: invalid input
Not an integer
Please enter process' priority, an integer: 2
Please enter the process' creation time (Hour:Minute): invalid input
Please enter the process' creation time (Hour:Minute): 12:12:12
Please enter the process' creation time (Hour:Minute): 14:00
Please enter if process is User mode, "True" or "False": invalid input
Not an acceptable boolean
Please enter if process is User mode, "True" or "False": False

Current process queue:
Process ID: 2446, Priority: 20
Process ID: 2546, Priority: 9
Process ID: 7481, Priority: 4
Process ID: 9999, Priority: 2

Create a process, "True" or "False":
```


The screenshot above shows the ability of our code to add new processes from user interface. As can be seen in the screenshot, only a process with a non-redundant ID can be created, and type checks are performed on each user-inputted parameter to ensure their validity. Further evident in the screenshot is the newly inputted user process being added to correct location in the process queue with respect to process priority.



```
Notes-MacBook-Pro:winch11 ndevine$ python3 PCB.py
Please enter the filename of the file you would like to read in: dataset.txt

--- READING FILE ---

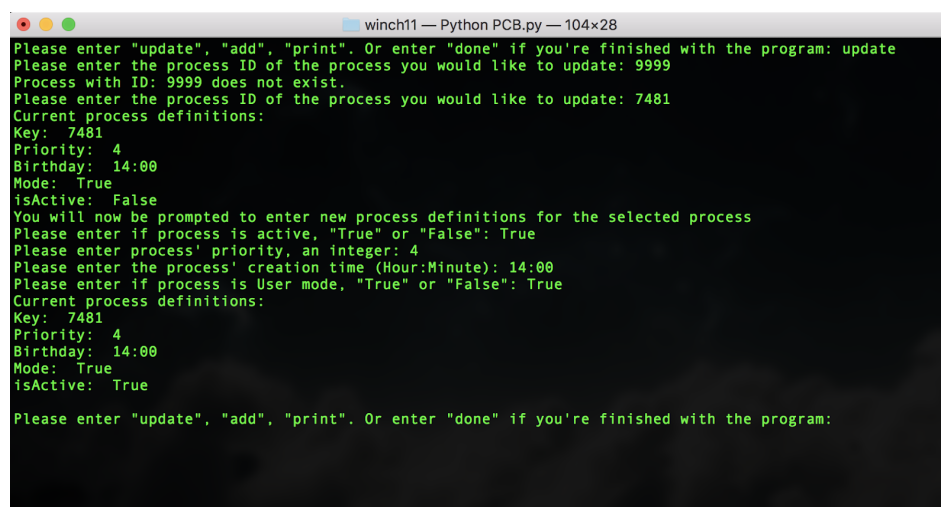
Not acceptable hours
Process ID 3546 isn't valid. Moving on to the next process...
Process ID 2446 has been validated
Process ID 2546 has been validated
Process ID 5154 isn't valid. Moving on to the next process...
Process ID 3135 isn't valid. Moving on to the next process...
Process ID 7481 has been validated

--- FINISHING READING FILE ---

You can add a new process, update a process, see active processes or finish with the program
Please enter "update", "add", "print". Or enter "done" if you're finished with the program: print
Of the processes you inputted, the active processes are: ['2446', '2546']
Of the processes you inputted, the inactive processes are: ['7481']

Please enter "update", "add", "print". Or enter "done" if you're finished with the program:[]
```

Above, this screenshot demonstrates the ability of our code to print out active/non-active processes. A processes activity state is determined by the second of the comma separated values in the provided .txt file. Accordingly, processes 2446 and 2546 are printed as active because they had an active value of True, while process 7481, though valid, is printed as inactive because it has an active value of False. The remaining processes are not printed because they are invalid.

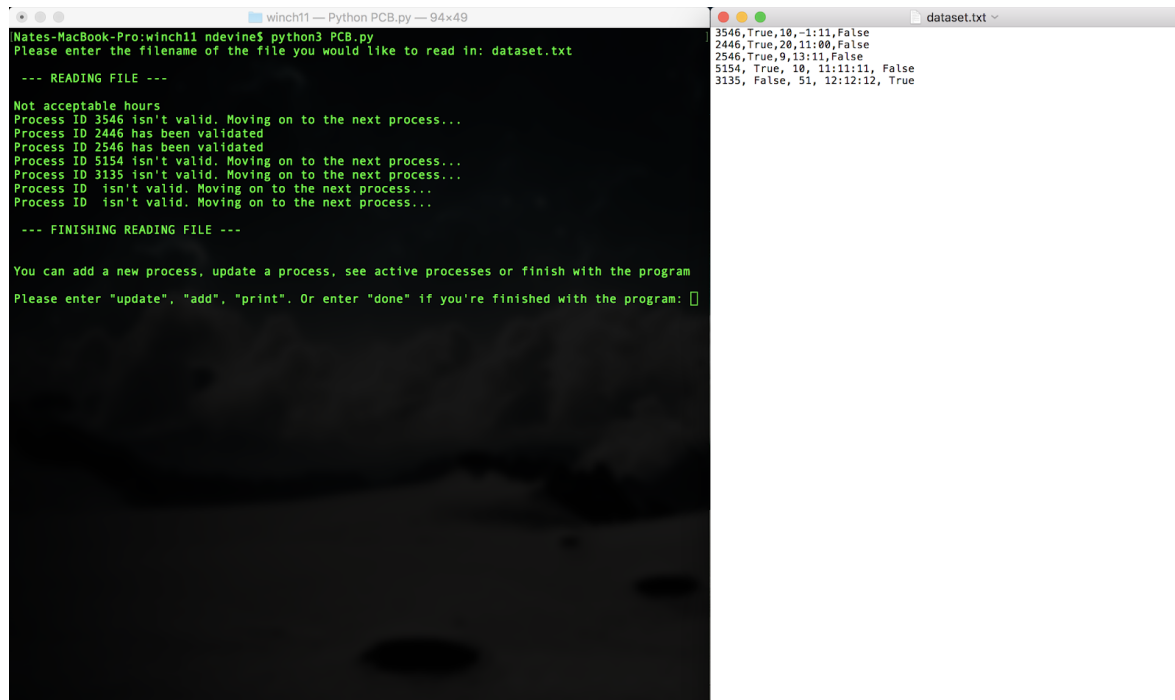


```
winch11 — Python PCB.py — 104x28

Please enter "update", "add", "print". Or enter "done" if you're finished with the program: update
Please enter the process ID of the process you would like to update: 9999
Process with ID: 9999 does not exist.
Please enter the process ID of the process you would like to update: 7481
Current process definitions:
Key: 7481
Priority: 4
Birthday: 14:00
Mode: True
isActive: False
You will now be prompted to enter new process definitions for the selected process
Please enter if process is active, "True" or "False": True
Please enter process' priority, an integer: 4
Please enter the process' creation time (Hour:Minute): 14:00
Please enter if process is User mode, "True" or "False": True
Current process definitions:
Key: 7481
Priority: 4
Birthday: 14:00
Mode: True
isActive: True

Please enter "update", "add", "print". Or enter "done" if you're finished with the program:
```

This screenshot above demonstrates our ability to update processes using the interface. As can be seen in the screenshot, the entered process ID must match a pre-existing process ID. Then, the user is given the option to update each parameter, and these updated selections are reflected when the process definitions are printed for a second time.



```
winch11 — Python PCB.py — 94x49
Nates-MacBook-Pro:winch11 ndevine$ python3 PCB.py
Please enter the filename of the file you would like to read in: dataset.txt

--- READING FILE ---
Not acceptable hours
Process ID 3546 isn't valid. Moving on to the next process...
Process ID 2446 has been validated
Process ID 2546 has been validated
Process ID 5154 isn't valid. Moving on to the next process...
Process ID 3135 isn't valid. Moving on to the next process...
Process ID  isn't valid. Moving on to the next process...
Process ID  isn't valid. Moving on to the next process...

--- FINISHING READING FILE ---

You can add a new process, update a process, see active processes or finish with the program
Please enter "update", "add", "print". Or enter "done" if you're finished with the program: 
```

```
dataset.txt ~
3546,True,10,-1:11,False
2446,True,20,11:00,False
2546,True,9,13:11,False
5154, True, 10, 11:11:11, False
3135, False, 51, 12:12:12, True
```

Above, this screenshot shows the ability of our code to validate the processes from the user's selected file. The process in the first line of the file is rejected because of invalid time format. The processes in the second and third lines are accepted as each parameter is valid and correct, the processes in the fourth and fifth lines are both rejected because of invalid time formatting. Finally, the final two processes are identified as invalid because they are simply blank lines from the text file.

Project Plan

Key components

Create an O/S Scheduler that allows for the simulation of several scheduling algorithms. It will be data-driven, have a User Interface and based on our team's chosen development technology. All of the other requirements will be derived by our team, via interviews/email with the CUSTOMER. Your ability correctly capture, document and implement the requirements as well as your enhancements will be key to your grade.

Roles and responsibilities.

- Project Manager (JALEEL WATLER) – coordinates the development effort, provides status updates on the effort to the customer. Manages a Project Plan.
 - Architect (KHÁNH NGHIÊM) – designs the high-level view of the key components in the solution, how they integrate and what they do, leads Software Design Document authoring. Present design to customer.
 - Software Designer (NATE DEVINE, ELIJAH PINEDA) – leads the coding effort, defines the data structures, classes and overall design of the code. Specifies testing approach. Presents how the solution will work to the customer.
 - Business Analyst (JOHNATHAN EVANILLA) – creates the requirements for the solution (both functional and non-function (so what it does as well as how it must work), customer facing for understanding the requirements. Helps in authoring requirements sections and other customer-centric areas of SDD.
 - Quality Tester (SAM BARNES) – ensure that the code works robustly, without errors. Fully test each function from each class, so that they do exactly what they are intended to do. Maintain the quality of the code.
-

Kickoff meeting

Kickoff meeting was held on Tuesday, October 2nd. We also used this time for team building by discussing things over the dinner table. The topics discussed during the kickoff meeting included deciding the roles and responsibilities of each team member. We also covered how the team will make decisions. We will first try to come to a decision as a whole and if the team is split on something, it will be put to a vote with 3/5 votes necessary for final decision. We also decided on some important ground rules:

Scope Statement.

The aim of this project will be to create an O/S Scheduler that allows for the simulation of several scheduling algorithms. It will be data-driven, have a User Interface and based on your team's chosen development technology. All of the other requirements will be derived by your team, via interviews/email with the CUSTOMER. Your ability to correctly capture, document and implement the requirements as well as your enhancements will be key to your grade.