**Pintos Project 1 : User Program (1)**

**(Project Report)**

Subject: :[CSE4070] Operating Systems

Faculty Member: Prof. Youngjae Kim

Group Number / Members: 53team, 20151562 신은우, 20151616 최승환

Development period: 2019/10/26 – 2019/11/01

**Project Title : Pintos Project 1 User Program (1)**

**Submission Date : 2019년 11월 1일**

**Members : 신은우, 최승환**

1. **Goal of Development**

* **Systematical, Cooperative Workflow**
  + This project follows workflow of ‘Agile Development’ technique. According to the changing situation, one should elastically change the plans accordingly, without losing consistency of the initial plan.
* **Stable, Reversible version control**
  + This project uses Github through Git commands. Throughout the whole project, turning points or potential danger zones should be noticed to each other teammate. Fatal errors should be prevented by commits and merges between the branches at the right moments, retaining valuable information about previous versions.
* **Functional Operating System**
  + This Operating System should be able to function all the basic functionalities this course suggests.   
    (total of 13)
* **Robust Operating System**
  + This Operating System should prevent predictable errors from various inputs and thread (or process) creation and deletions  
    (total of 8)

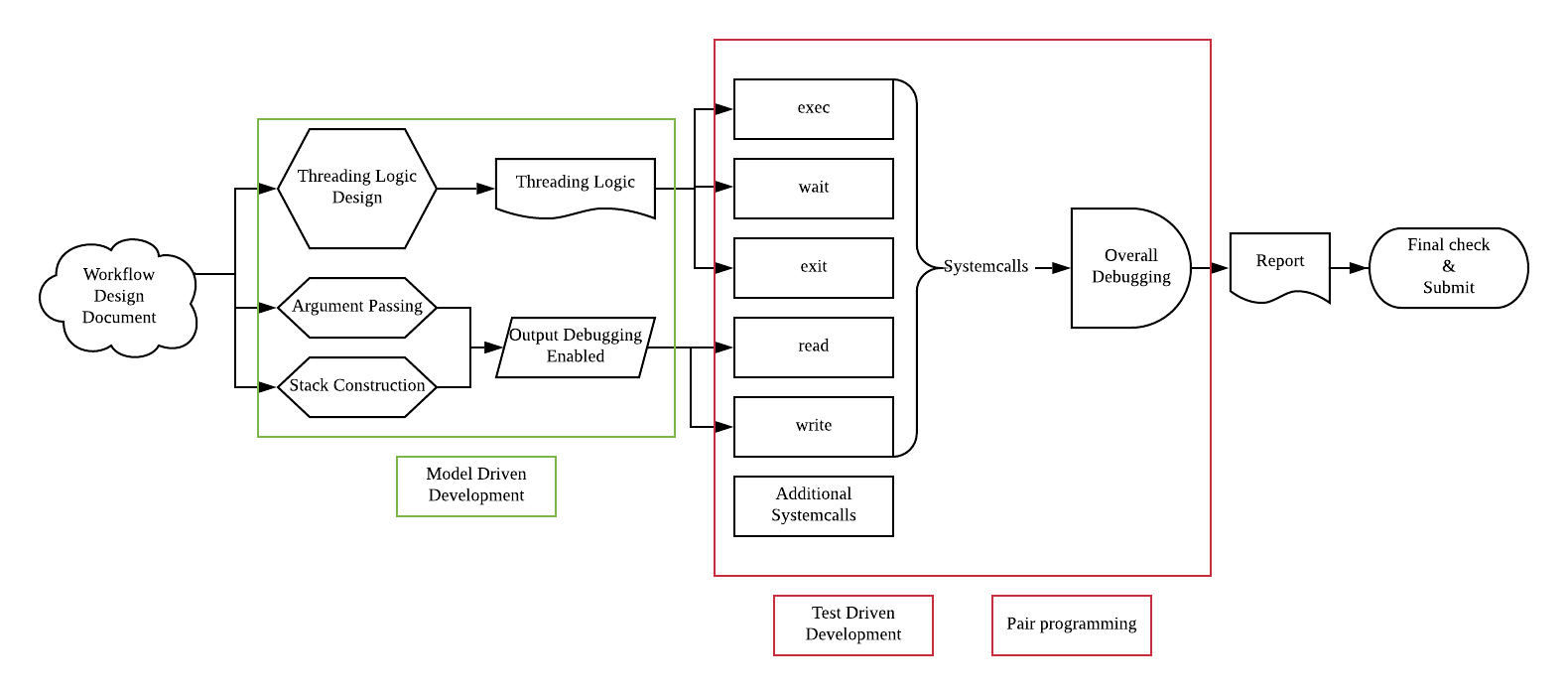
1. **Range and Contents of Development** 
   1. **Range of Development**

* **Argument Passing :** distinguish the parameters put into the memory, and construct a stack according to the format of the stack
* **User Memory Access :** Check the input’s memory access. Confirm its validity compared to the user memory access range
* **System Calls** 
  + **HALT :** terminate pintos by shutdown\_power\_off()
  + **EXIT :** exit from the process, returning exit status by child process.
  + **WAIT :** wait for the child process to finish. After that, returns the exit status of the child process
  + **EXEC :** runs the code according to input in command line. passes given arguments, returns new process pid
  + **READ :** Read from standard input(keyboard).
  + **WRITE :** Write to console.
  + **FIBONACCI :** return the fibonacci function value of input number.
  + **SUM\_OF\_FOUR\_INT :** return the sum of four integer inputs from command line
  1. **Contents of Development**
* **Page**
  + Page is the region of the virtual memory which is granted by OS to the Process for use. The ceiling of the page’s address value is limited by PHYS\_BASE which is 0xc0000000 in this case of pintos. Virtual memory is evenly divided by page size. and page number combined with page offset consists the virtual address
* **Process & Thread**
  + Process is a program continuously being run in a computer, using a specific individual memory space assigned by OS. All process has at least one thread, which becomes the unit of several code flow in the process. Since Pintos is a single threaded system, one process is equal to a single thread. Each thread has a even time quantum and is scheduled by the ‘time scheduling algorithm of the OS.
* **User memory accessing** 
  + User process is a process executed by the user. Kernel memory is a memory region which is used for operations of OS. Since OS is responsible for various jobs including I/O operations, memory corruption from the user process can cause a variety of problems. To prevent these errors, several methods are used. In case of this project, user memory is always assigned under 0xc0000000. Therefore always check whether the pointer value passed by the argument exceeds 0xc0000000. Also check if the memory access is the access to a valid memory address by checking if it’s in the right page assigned to the user

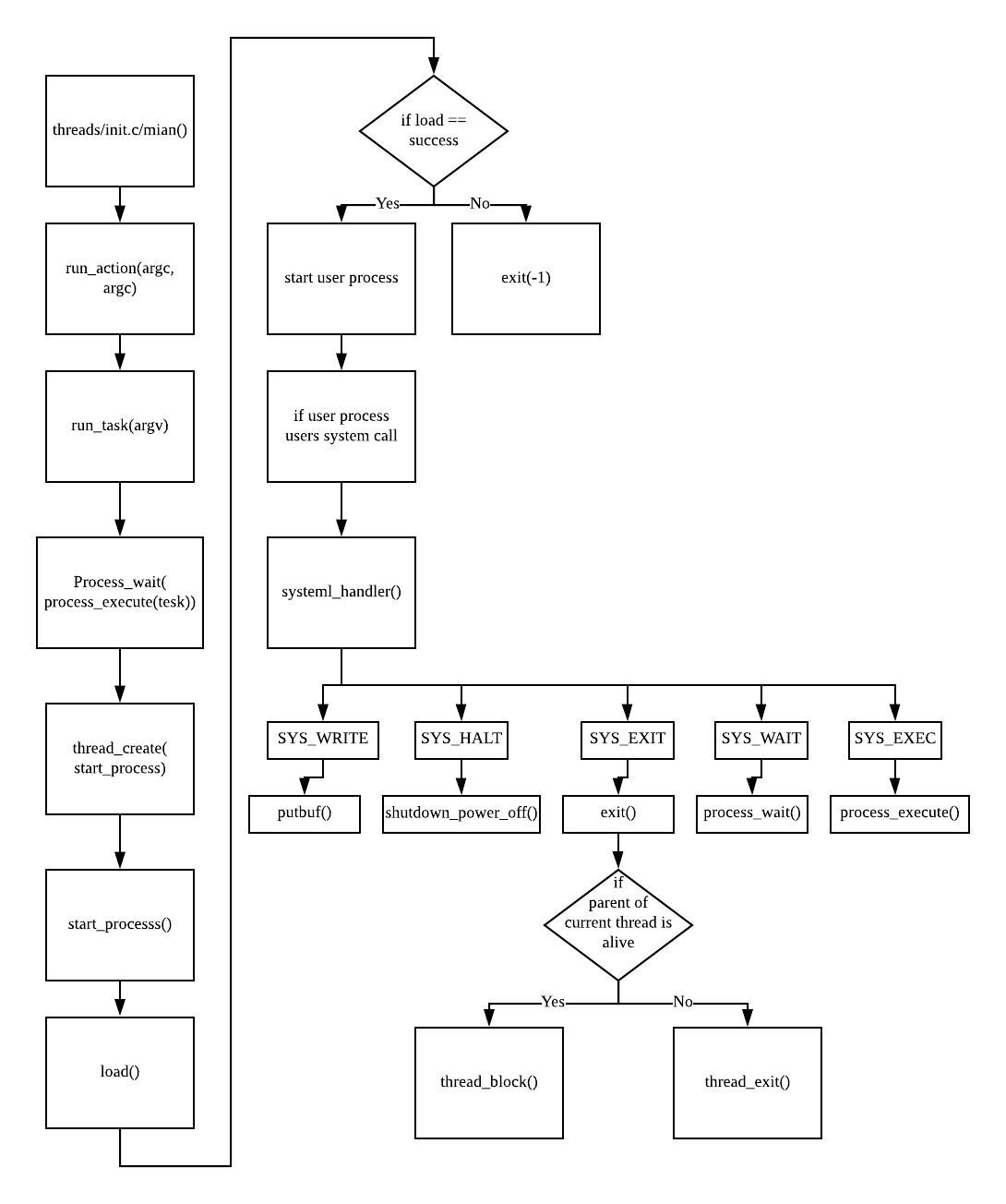
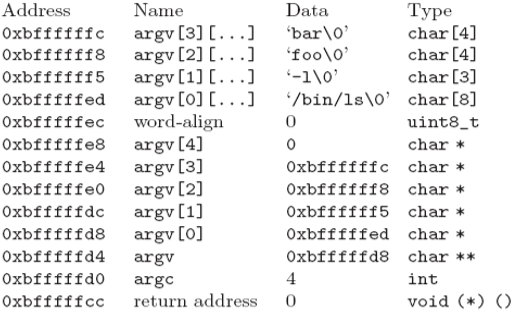
1. **Schedule and Methodology of Development** 
   1. **Schedule**

* **10.26 ~ 10.28** Argument passing + Stack construction **10.26 ~ 10.30** Thread Logic Design + implementation **10.31 ~ 11.1** Systemcalls & debugging (TDD) **11.1** Report + final check
  1. **Methodology**
* **Agile Development  
  (Model Driven Development, Test Driven Development + Pair Programming)**
  1. **Role Sharing**
* **Argument passing, Stack construction** - 최승환
* **Threading** - 신은우
* **Systemcall wait, exec, exit** - 신은우
* **Systemcall read, write** - 최승환, 신은우
* **Additional systemcall** - 최승환
* **Overall Debugging** - 신은우, 최승환

**D. Devleopment Flow**

* ****

1. **Result of Research** 
   1. **Composition of Contents**

* ****
  1. **Development of Contents**
* **Argument Passing**
  + Parsing input : process\_execute() runs the program, however does not work in a intended manner if the parameters are not saved in the memory. To do that, the first thing to consider is ‘what part of input means what value?’. To meet the following question, input should be divided by given rule ,in this project [spacebar]. In this project, we divide the input by ‘ ‘. So, we use strtok\_r() to take ‘ ‘ as a parameter value. After that, use the next\_token continuously in a loop to parse all words in the input until the input ends. Each word then will be saved in (char\*)token, eventually saved in (char\* []) input\_words for later use.
  + Constructing the stack : words parsed by the above operation is saved (char\* [])in input\_words. for all these words in input\_words, its value, and its saved address(pointer) should be in the stack. Following picture is the order of the variables saved in stack.  
      
    interesting values will be explained  
    word-align : used to fix the access address by multiplication of four.   
    argv[i] : address of the argv[i] and values are saved  
    argc : number of variables passed  
    return address : the return address to where process should direct after termination
* **User Memory Access**
  + On writing the Systemcalls, arguments are passed by the user process to the systemcall function. By doing so, kernel should check if the user process passed the right parameters or not. When given arguments were wrong, serious problems can occur.  
     is\_user\_vaddr() function takes pointer as argument, and confirms if that pointer is a user memory address or not. pagedir\_get\_page() checks if certain page is assigned address. Using these two function, OS identifies the argument of the user process. First, using the esp from the user, is\_user\_vaddr() checks the argument. Since arguments are pushed to esp, check if esp is in the user memory range. The argument address also should be checked. In the process, use both is\_user\_vaddr() and pagedir\_get\_page() to see if the address is right.
* **System Call**
  + **halt** : Terminates Pintos by calling shutdown\_power\_off() (declared in ‘devices/shutdown.h’).
  + **exit & wait & exec**
    - exit, wait, exec part does not simply terminate, wait , and execute but also manages what thread is parent thread and what thread is child thread. This management is implemented by ‘struct Family’ and ‘struct Child’ illustrated below.
    - ‘struct Family’ is a structure to maintain the relation between Parent and Child. Each Family is implemented by pintos doubly linked list. Each family element manages its own tid, parent’s tid, child’s tid. All child tid is managed by ‘struct list’ of child. child list also is implemented through doubly linked list and saves its own state, its own tid.
    - **Struct Code**

enum ChildStatus {

CHILD\_ALIVE, CHILD\_DIE, CHILD\_KILL, CHILD\_READY

}

struct Child {

tid\_t tid;

enum ChildStatus status;

struct list\_elem elem;

int exitvalue;

}

struct Family {

tid\_t parent;

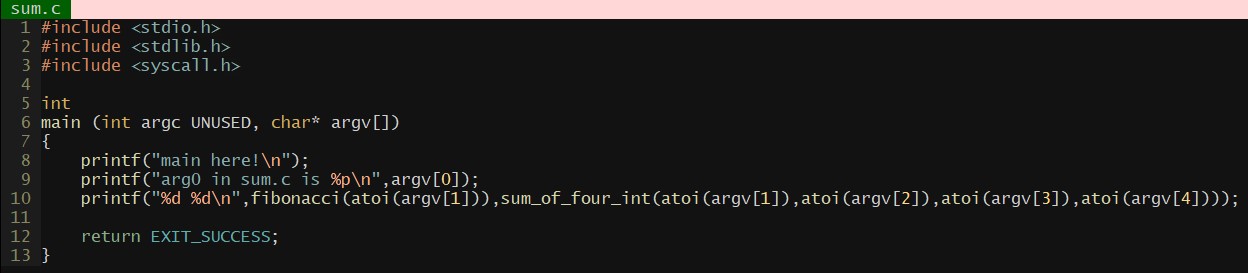
tid\_t me;

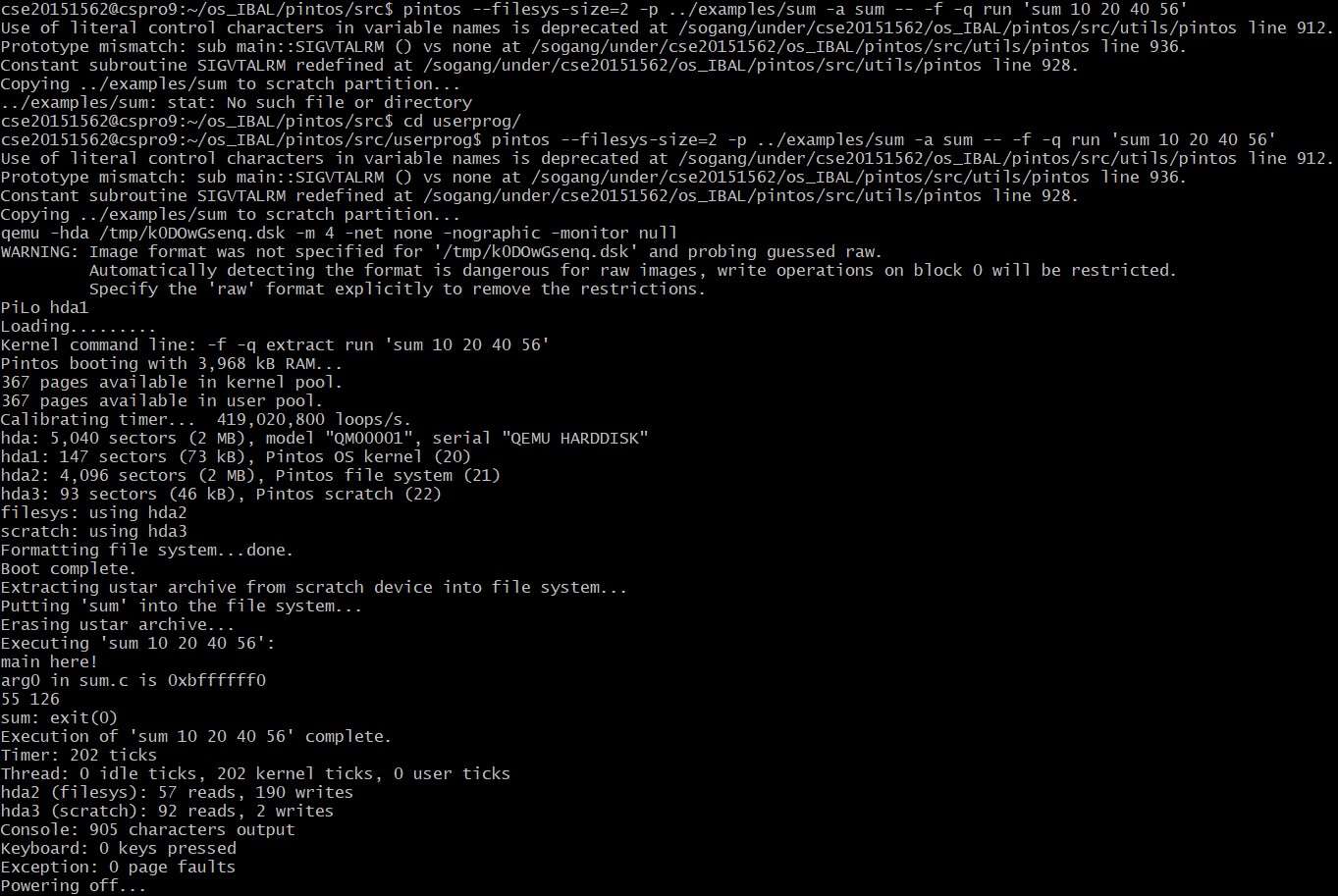
struct list child\_list;

struct list\_elem elem;

}

* + - **Flow**
      * When Exec System call is called, process\_execute() is called. Then process\_execute() calls thread\_create(). thread\_create() creates a thread and puts it into a ready queue and puts the thread into family list, by makeFamily().  
         makeFamily() pushes the Child struct into the child list of the parent. until process\_execute() returns the tid, familyCheckChildState checks the state of the child by child’s tid.(busy wait) If its child thread is not executed in a normal way, returns -1. Otherwise returns the child’s tid
      * Exit systemcall calls familyChildToDie() to save its status to ‘die’ and thread\_exit() to terminate. thread\_exit() uses familyKillMe() to remove self from family list. If the process was terminated in a different way. save its state in the child list as ‘kill’ (abnormal terminate) rather than ‘die’(normal terminate)
      * When wait system call is called, process\_wait() calls familyCheckChildState() to busy wait, and familyDeleteChild() to remove child process from child list. If child was not terminated in a normal way. If child’s tid does not exist, return -1. Otherwise return the child process’s return value.
  + **read**
    - use for loop to get single character n times. input\_getc() takes single character input and returns it from stdin
  + **write**
    - use putbuf() to print the arguments on stdout
* **Additional System Call**
  + **Fibonacci**
    - use three variable old, new, temp to calculate next fibonacci value. Dynamic Programming lets this process to finish in O(N).
  + **SUM\_OF\_FOUR\_INT**
    - Adds four integer parameters and return the sum of the four
  1. **Test and Evaluation of Contents**

****

****

1. **Etc.** 
   1. **Level of Contribution**

* **신은우 (50%)  
  최승환 (50%)**
  1. **Thoughts for Project**
* **최승환** 
  + Whole job from start to the end was tightly organized and divided. Even though there were some delays on some parts, the jobs were done step by step according to the workflow.  
     As much effort as we put on the implementation, we took a close look of the whole project jobs and its relations. We tried to fit each module in the right place at right time.  
     Also used github to save previous versions and make branches to distinguish and merge the jobs each teammate is doing. I am quite happy about the process of the work we went through as much as the good results.
* **신은우**
  + Ever since I took the class to understand the codes, I didn’t fully understand the detailed concepts of the OS. It was a good experience for me to reconstruct the details through writing it into a full code by myself.  
     After the start of the project, I couldn’t understand the code flow well, so I looked for the contents on the Pintos manual and project PPT slides. It was a hard job even after reading those material. After I figured it out step by step, I could comprehend the knowledge about operating system. The accomplishment after completing the code gave me confidence to proceed succeeding projects.