```
+---------------------------+
|          CS 140           |
|  PROJECT 2: USER PROGRAMS |
|     DESIGN DOCUMENT       |
+---------------------------+
```

---- **GROUP 9** ----

Aditya Singh <aditya390402@gmail.com>
Itish Agarwal <itishagarwal2000@gmail.com>

---- **PRELIMINARIES** ----

>> References
https://web.stanford.edu/class/cs140/projects/pintos/pintos_3.html
https://www.youtube.com/watch?v=OE79vNZp1KI&t=1239s
http://bits.usc.edu/cs350/assignments/project2.pdf
https://inst.eecs.berkeley.edu/~cs162/sp16/static/projects/project2.pdf

ARGUMENT PASSING
===========

---- **DATA STRUCTURES** ----

>> A1: Copy here the declaration of each new or changed `struct' or `struct' member, global or static variable, `typedef', or enumeration. Identify the purpose of each in 25 words or less.

For argument passing, no extra ‘struct’ or global variable was declared. We took care of argument passing in process.c, where we defined a new function get_stack_args(char *file_name, void **esp, char **tok_ptr), which is used to add arguments to the stack and add padding and alignment if needed and ultimately update the stack pointer(esp).

---- **ALGORITHMS** ----

>> A2: Briefly describe how you implemented argument parsing. How do you arrange for the elements of argv[] to be in the right order? How do you avoid overflowing the stack page?

The raw pintos comes with no feature to accommodate command line arguments passed when running the executable. Any process (whether kernel or user's) directly or indirectly calls process_execute. Inside process_execute, start_process is called which in turn calls function load, which pushes the arguments on the stack.
As the load() function is used to load files we also load values in the stack. This is done by calling a function 'setup_stack()' which is used to add values to the stack. If the stack is successfully set we then call the function 'get_stack_args()' where we add arguments to the stack.
We then take the variable 'total_length' which will store the total length of the arguments. We take a loop and subtract the required space for arguments. We also maintain a variable to keep a count of the number of arguments(ie, 'argc'). We then save the current position of the stack in a variable.
After this, null character is added and esp is updated by subtracting the sizeof(char*).
In this way addresses of the arguments are added. After adding addresses we add a pointer to the first address and update the esp. At last we update the stack pointer and add a fake return address. After adding the arguments we maintained the stack as a multiple of 4 (called word alignment).

We avoid overflowing the stack by keeping a check on the total size of the arguments being passed. If at any point the size of arguments exceeds the size of stack, we exit.

---- **RATIONALE** ----

>> A3: Why does Pintos implement strtok_r() but not strtok()?

In strtok_r(), the placeholder is provided by the caller, unlike strtok(). Since PintOS separates commands into executable names and arguments via the kernel, we need to store the address of the arguments to an accessible place so that we can make sure that the arguments don’t get mixed up when multiple threads call strtok_r(). Each thread has a pointer (tok_ptr), independent from the caller, which is a determiner for its position.

>> A4: In Pintos, the kernel separates commands into an executable name and arguments. In Unix-like systems, the shell does this separation. Identify at least two advantages of the Unix approach.

The advantages of the Unix approach are as follows:

1. Input validation is much safer if done by the shell instead of the kernel. A problem could be caused if the kernel failed to parse data (for example, a large amount of text is passed as input) whereas in shell, the worst case scenario is that it simply crashes.

2. Separating the commands from the arguments can allow for some input pre-processing and is generally a cleaner approach since commands and arguments are separate from each other. Plus, there’s no reason for a kernel to parse data, which can be easily done by a user program.