|  |  |
| --- | --- |
|  | +----------------------------------+ |
|  | | | |
|  | | PROJECT 2: USER PROGRAMS | |
|  | | DESIGN DOCUMENT | |
|  | +----------------------------------+ |
|  |  |
|  |  |
|  |  |
|  | ---- GROUP 9 ---  Krisha Rain Martinez [rainmartinez@gmail.com](mailto:rainmartinez@gmail.com)  Jericho Mamaradlo [mamaradlojn@gmail.com](mailto:mamaradlojn@gmail.com)  Rod Labarete [agarchnash08@gmail.com](mailto:agarchnash08@gmail.com) |
|  |  |
|  |  |
|  |  |
|  | ---- PRELIMINARIES ---- |
|  |  |
|  | >> If you have any preliminary comments on your submission, notes for the |
|  | >> TAs, or extra credit, please give them here. |
|  |  |
|  | >> Please cite any offline or online sources you consulted while |
|  | >> preparing your submission, other than the Pintos documentation, course |
|  | >> text, lecture notes, and course staff.  https://web.stanford.edu/class/cs140/projects/pintos/pintos\_3.html#SEC32 |
|  |  |
|  | 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. |
|  |  |
|  |  |
|  |  |
|  | ---- 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? |
|  |  |
|  | How to implement argument parsing? |
|  | ---------------------------------- |
|  | The most important part was to setup the stack. We did it inside setup\_stack () |
|  | after page is installed, when the stack has been initialized. |
|  |  |
|  | Process\_execute provides file\_name, including command and arguments |
|  | string. First, we separated the first token and the rest, which are command and |
|  | arguments. We use command as the new thread's name, and pass down the arguments |
|  | string to start\_process(), load() and setup\_stack(). We think it’s implementable |
|  | since we can always get the command name from thread->name when needed, like |
|  | when load the ELF executable. |
|  |  |
|  | When setting up the stack, we memcpy the argument string and then the command |
|  | name which is actually the thread name in our case. Then add alignment, scan the |
|  | string backward to get each token and push its address into the page underneath |
|  | the alignment to generate argv[], finally argv, argc and return address. |
|  |  |
|  | Way of arranging for the elements of argv[] to be in the right order. |
|  | -------------------------------------------------------------------- |
|  | We scan through the argument string backwards, so that the first token we get is |
|  | the last argument, the last token we get is the first argument. We can just keep |
|  | decreasing esp pointer to setup the argv[] elements. |
|  |  |
|  | How to avoid overflowing the stack page? |
|  | ---------------------------------------- |
|  | The thing is we decided not to check the esp pointer until it fails. Our |
|  | implementation didn’t pre-count how much space do we need, just go through |
|  | everything, make the change, like add another argv element, when necessary. But |
|  | this leaves us two way to deal with overflowing, one is checking esp’s validity |
|  | every time before use it, the other one is letting it fails, and we handle it in |
|  | the page fault exception, which is exit(-1) the running thread whenever the |
|  | address is invalid. We chose the latter approach since the first approach seems |
|  | have too much burden and it make sense to terminate the process if it provides |
|  | too much arguments. |
|  |  |
|  | ---- RATIONALE ---- |
|  |  |
|  | >> A3: Why does Pintos implement strtok\_r() but not strtok()? |
|  |  |
|  | The only difference between strtok\_r() and strtok() is that the save\_ptr |
|  | (placeholder) in strtok\_r() is provided by the caller. In pintos, the kernel |
|  | separates commands into command line (executable name) and arguments. So we need |
|  | to put the address of the arguments somewhere we can reach later. |
|  |  |
|  | >> A4: In Pintos, the kernel separates commands into a executable name |
|  | >> and arguments. In Unix-like systems, the shell does this |
|  | >> separation. Identify at least two advantages of the Unix approach. |
|  |  |
|  | 1) Shortening the time inside kernel |
|  | 2) Robust checking. Checking whether the executable is there before passing it |
|  | to kernel to avoid kernel fail. Checking whether the arguments are over the |
|  | limit. |
|  | 3) Once it can separate the commands, it can do advanced pre-processing, acting |
|  | more like an interpreter not only an interface. Like passing more than 1 set |
|  | of command line at a time, i.e. cd; mkdir tmp; touch test; and pipe. |

**USER MANUAL**

# How to run Pintos on your machine?

$> cd/pintos/userprog

$> make

$> cd build

$> pintos

# Make functions:

$> make check (it will test your solution against a set of test programs. Make sure you 'make clean' between runs.)

# Test results:

