```
+--------------------------+
| CS 421/521               |
| PROJECT 2: USER PROGRAMS  |
| DESIGN DOCUMENT          |
+--------------------------+
```

---- GROUP ----

>> Fill in the names and email addresses of your group members.

. Manvijay <manvijay@buffalo.edu >
Elwin Cabrera <elwincab@buffalo.edu >
Yang Xue < yxue5@buffalo.edu >

---- PRELIMINARIES ----

>> If you have any preliminary comments on your submission, notes for the
>> TAs, or extra credit, please give them here.

>> Describe briefly which parts of the assignment were implemented by
>> each member of your team. If some team members contributed significantly
>> more or less than others (e.g. 2x), indicate that here.

FirstName LastName: contribution
FirstName LastName: contribution
FirstName LastName: contribution

>> 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.

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.

We haven't declared any new or changed any `struct' or
`struct' member, global or static variable, `typedef', or
enumeration for the purpose of implementing Argument Passing

---- 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 function stack_frame_size calculates the amount of size the stack would require and adds padding to avoid overflow. For putting the elements of argv[] in the right order we used strtok_r and iterated over the string file_name and for every iteration we get a token back and the length of that token is used to calculate how much do we need decrement the value stored in esp, *esp-= strln(token)+1. Then we save the value of esp in argv[argc]. And keep incrementing argc.

---- RATIONALE ----

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

The reason Pintos use strtok_r() and not strtok() is because strtok_r() is safe to use with multiple threads running at once as it saves the context of the next token separately. Put in simple words strtok_r() is re-entrant version of strtok().

>> 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.

As the separation is being done on the user side, operations at kernel side become simpler.

Also, this helps in reducing the number of potential bugs in kernel as it restricts user from passing bad arguments.

Shell also allows better resolution of relative paths as it understand the environment better to resolve the paths.

SYSTEM CALLS

---- DATA STRUCTURES ----

>> B1: 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.

We haven't declared all the variables that we will need for implementing system calls yet but we will use one struct to keep the information about the file and its structure is as follows:

```
struct file_descriptor {
/* the unique file descriptor number */
int file_descriptor_num;
/* file that is opened */
struct file *file_struct;
/* the owner thread's thread id of the open file */
tid_t owner_thread;
};
```

Then we will also use a list of open files which is declared as follows:

```
Struct list open_files_list;
```

>> B2: Describe how file descriptors are associated with open files.
>> Are file descriptors unique within the entire OS or just within a
>> single process?

File descriptors are the number entries for every open file. File descriptors will be unique within entire OS to prevent any kind of confusion.

---- ALGORITHMS ----

>> B3: Describe your code for reading and writing user data from the
>> kernel.

To be decided.

>> B4: Suppose a system call causes a full page (4,096 bytes) of data
>> to be copied from user space into the kernel. What is the least
>> and the greatest possible number of inspections of the page table
>> (e.g. calls to pagedir_get_page()) that might result? What about
>> for a system call that only copies 2 bytes of data? Is there room
>> for improvement in these numbers, and how much?

If we assume that the memory is contiguous then for both the cases the best case scenario will be One Inspection (call to pagedir_get_page())and the worst case will be Two Inspections. However, if the memory isn't contiguous then worst case scenario of a full_page might be 4096 inspection and 2 inspections for the 2 bytes of data.

We don't see how we can improve on this from the algorithms we are familiar with.

>> B5: Briefly describe your implementation of the "wait" system call
>> and how it interacts with process termination.

To be decided.

>> B6: Any access to user program memory at a user-specified address
>> can fail due to a bad pointer value. Such accesses must cause the
>> process to be terminated. System calls are fraught with such
>> accesses, e.g. a "write" system call requires reading the system
>> call number from the user stack, then each of the call's three
>> arguments, then an arbitrary amount of user memory, and any of
>> these can fail at any point. This poses a design and
>> error-handling problem: how do you best avoid obscuring the primary
>> function of code in a morass of error-handling? Furthermore, when
>> an error is detected, how do you ensure that all temporarily
>> allocated resources (locks, buffers, etc.) are freed? In a few
>> paragraphs, describe the strategy or strategies you adopted for
>> managing these issues. Give an example.

To be decided.

---- SYNCHRONIZATION ----

>> B7: The "exec" system call returns -1 if loading the new executable
>> fails, so it cannot return before the new executable has completed
>> loading. How does your code ensure this? How is the load
>> success/failure status passed back to the thread that calls "exec"?

We haven't implemented this yet but this is what we might do. Once we have the child process, we can try to to load a new process. If the load is successful, we will set the status variable to 0 or -1 and return the status of load while returning.

>> B8: Consider parent process P with child process C. How do you
>> ensure proper synchronization and avoid race conditions when P
>> calls wait(C) before C exits? After C exits? How do you ensure
>> that all resources are freed in each case? How about when P

>> terminates without waiting, before C exits? After C exits? Are
>> there any special cases?

---- RATIONALE ----

>> B9: Why did you choose to implement access to user memory from the
>> kernel in the way that you did?

To be decided.

>> B10: What advantages or disadvantages can you see to your design
>> for file descriptors?

Advantages:

1. **We have our own struct file_descriptor which helps in minimizing the size of struct thread hence minimizing complexity in the code.**
2. **We are keeping track of all open files which gives us the ability to manipulate them whenever required.**

Disadvantages:
We can't think of any disadvantage at this point other than the overhead of keeping track of all the open files.

>> B11: The default tid_t to pid_t mapping is the identity mapping.
>> If you changed it, what advantages are there to your approach?

The default mapping seems implementable and we might stick to that.

SURVEY QUESTIONS
==========

Answering these questions is optional, but it will help us improve the course in future quarters. Feel free to tell us anything you want--these questions are just to spur your thoughts. You may also choose to respond anonymously in the course evaluations at the end of the quarter.

>> In your opinion, was this assignment, or any one of the three problems
>> in it, too easy or too hard? Did it take too long or too little time?

>> Did you find that working on a particular part of the assignment gave
>> you greater insight into some aspect of OS design?

>> Is there some particular fact or hint we should give students in
>> future quarters to help them solve the problems? Conversely, did you
>> find any of our guidance to be misleading?

>> Do you have any suggestions for the TAs to more effectively assist
>> students, either for future quarters or the remaining projects?

>> Any other comments?