CS 342 Design Document
Project 2A – Setting up stack for user program with arguments

Rajat Kateja (r.kateja@iitg.ernet.in)
Maths and Computing
10012338

TA: Lalatendu Behera
Chetti Prasad

------ DATA STRUCTURES ------
No new data structures were introduced for this project.

------ ALGORITHM ------

The algorithm used to set up the stack is the baisc algorithm of how a stack is setup for a userprogram with arguments.

Firstly the stack pointer is rounded down by a factor of 4, as suggessted in the manual, so as to increase speed. Then the arguments are pushed onto the stack in the form of strings. Then an integer value of 0 is pushed onto the stack. Then a NULL pointer. After this, the pointer to the argument string are pushed onto the stack in left to right order. Finally the pointer to the last pointer (amongst the ones just pushed) is pushed onto the stack, and then the total number of arguments to the user program are pushed. Finally for consistency, a fake return address is also pushed onto the stack so that the user program has a stack similar to any arbitrary function.

------ SYNCHRONISATION ------

Pushing onto the stack did not require any special synchronisation because each thread has its own user stack, and hence there are no race conditions to take care of.

------ RATIONALE ------

Setting up the stack in the format provided in the manual is a standard way of implementing user program argument passing. The sequence of the data elements pushed into the stack make it easy for the user program to access its arguments. For example, firstly it encounters the total number of arguments, which is then used to access only the required number of memory addresses.

The stack setting up has been done in the setup_stack function, which receieves an argument as the pointer to the stack pointer. Which means the the code to perform the stack setup is written is lots of dereferencing operators which reduces its readability a bit. The same could have been done in the start_process function, which would have improved readability, but it is semantically cleaner to setup the stack in the setup_stack function, because that is what it is meant to do.

------ SNIPPETS OF CODE MODIFIED ------

In the file /pintos/src/userprog/process.c

Two major modifications are parsing the string contating the file name along with the arguments and setting up the stack. Both of them are presented below:

Parsing:

```
char *token, *save_ptr;
char **args;
int i=0, count=0;
args = (char **)malloc(sizeof(char *));
for (token = strtok_r (file_name, " ", &save_ptr); token != NULL; token = strtok_r (NULL, "
                                                                    ", &save_ptr))
{
        *args = (char *)malloc(sizeof(char));
        *args = token;
        //printf("\n%s\n", *args);
        args++;
        i++;
        count++;
}
*args = NULL;
while(i>0)
{
        args--;
        i--;
}
```

Setting the stack:

```
*esp = PHYS_BASE - 4;
int i, count = 0;
for(i=0, count=0;file_args[i]!=NULL;i++, count++)
{}
char *add[count];
for(i=count-1;i>=0;i--)
{
        add[count-i-1] = *esp;
        *esp = *esp - strlen(file_args[i]) - 1;
        strlcpy(*esp, file_args[i], strlen(file_args[i])+1);
}
                *esp-=4;
                ** (int **) esp = 0;
*esp-=4;
                ** (char ***) esp = NULL;
for(i=0;i<count;i++)
{
        *esp = *esp-4;
        ** (char ***) esp = add[i];
}
char **zero_add;
zero_add = *esp;
*esp-=4;
** (char ****) esp = zero_add;
*esp -= sizeof(int);
```

```
** (int **) esp = count;
*esp -= sizeof(void *);
** (int **) esp = 0;
```