CSCI3150 – Tutorial 3

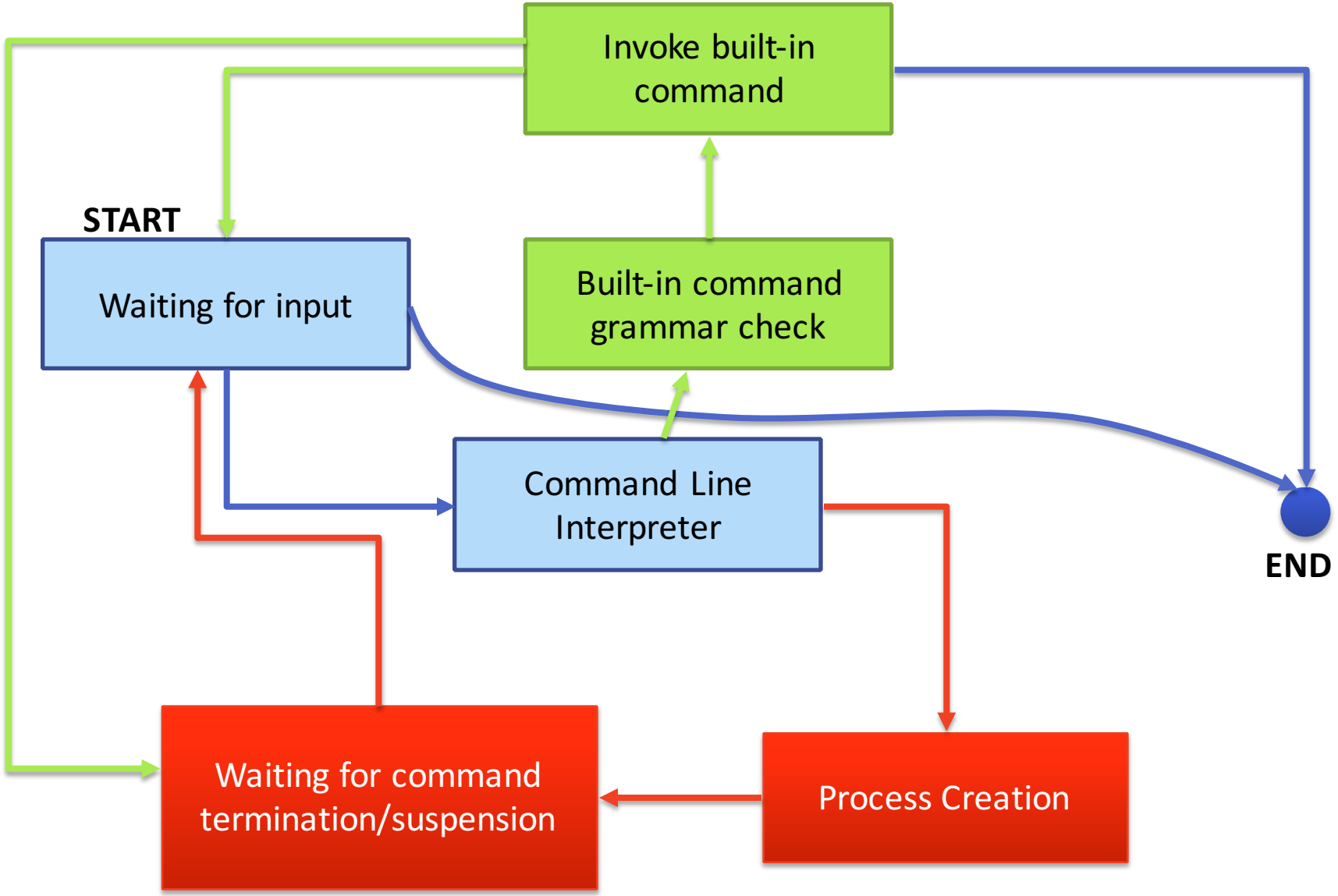
ASSIGNMENT 1, PHASE I

Calvin Kam <hckam@cse.cuhk.edu.hk>

What will you need to do

1. Phase 1 - Simple Shell environment

1. Phase 2 - A simple scheduler. 


Invoke built-in command
START
Waiting for input
Built-in command grammar check
Command Line Interpreter
END
Waiting for command termination/suspension
Process Creation

Get the User Input

<stdio.h>

```
[3150 Shell:/home/hckam]$ ls -al
```

- Get the whole input including spaces…. “ls -al“ whether than “ls” alone.
- `fgets()` is recommended, as it is easy to use and get all the things you need.

```c
#include <stdio.h>
#include <string.h>

int main(int argc,char *argv[]){
    char buf[255];
    printf("[I am Shell]$:");
    fgets(buf,255,stdin);
    printf("%s\n",buf);
    return 0;
}
```

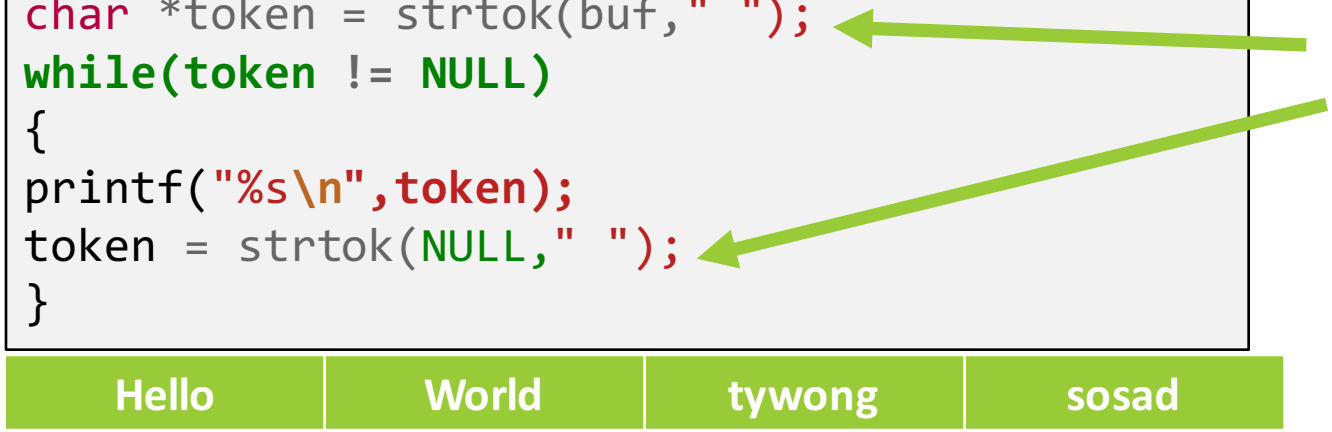
Tokenize the input

<string.h>

After you get the input, the string is "chopped" into words (tokens) in order to be examined (for example, check whether it is a built-in command) and build the argument list.

A useful string function - `char *strtok(char *str, const char *delim);`

```
#include <string.h> //Needed by strtok()
char buf[] = "Hello World tywong sosad";
char *token = strtok(buf," ");
while(token != NULL)
{
printf("%s\n",token);
token = strtok(NULL," ");
}
```



Get Current Directory

<unistd.h>
<limits.h>

In the prompt, we need to know the current working directory.

Initially the path is where the program is located.

The useful function is - `char *getcwd(char *buf, size_t size);`

PATH_MAX is the maximum characters allowed in the pathname.

```
#include <stdio.h>
#include <limits.h> // Needed by PATH_MAX
#include <unistd.h> // Needed by getcwd()

int main(int argc,char *argv[]){
char cwd[PATH_MAX+1];
if(getcwd(cwd,PATH_MAX+1) != NULL){
printf("Current Working Dir: %s\n",cwd);
}
else{
printf("Error Occured!\n");
}
return 0;
}
```

Change the working directory [cd]

<unistd.h>

It is a **built-in** command and it changes the current working directory.

The useful system call is `int chdir(const char *path);`

```
char buf[PATH_MAX+1];
Char input[255];
if(getcwd(buf,PATH_MAX+1) != NULL){
    printf("Now it is %s\n",buf);
    printf("Where do you want to go?:");
    fgets(input,255,stdin);
if(chdir(input) != -1){
    getcwd(buf,PATH_MAX+1);
    printf("Now it is %s\n",buf);}
else{
        printf("Cannot Change Directory\n");
}
```

Any Problem?

```
Where do you want to go: /tmp
Cannot Change Directory
```

- Debug Skills

<string.h>

- In order to know the details of the error (For assignment you don't need to do so in some cases), we can check the variable "errno"
- "errno" is a set of **numbers** set by system calls and some library functions in the event of error.
- We can use strerror() to return the string description of the error number.

```
if(chdir("/tmpp") != -1){
getcwd(buf,PATH_MAX+1);
printf("Now it is %s\n",buf);
}
else{
printf("Cannot Change Directory\n");
// For Debug purpose, no need detail messages in the assignment
printf("Error is %d,[%s]\n",errno,strerror(errno));
}
```

Remove '\n' at the end

fgets() will get the input together with the '\n' (Carriage return).

We need to **remove the ending '\n'** for the correct directory name.

```
fgets(input,255,stdin);
input[strlen(input)-1] = '\0';
```

This turns the ending '\n' to the terminating character '\0'.

`strlen(input) = 3`

0	1	2	3
l	s	\n	\0
l	s	\0	\0

Process Creation

<unistd.h>

The shell is now ready to run the command!

To create a new process in C, use the system call `pid_t fork(void)`

The parent returns value > 0 and the child returns 0.

```
if(fork()){
        printf("I am Parent,my pid is %d\n",getpid());
}
else{
        printf("I am Child, my pid is %d\n",getpid());
}
```

Wait…Why do I need to create a new process?

From the lecture we learnt that `exec()` system call family can execute other program. 👍

Why do we need call `fork()`?

```c
#include <stdio.h>
#include <unistd.h> // Needed By execvp

int main(int argc,char *argv[])
{
        printf("Calling ls...\n");
        char *arglist[] = {"ls",NULL};
        execvp(*arglist,arglist);

        printf("Will the girl 'ls' reply my message?\n");
        return 0;
}
```

exec() system call never returns if success.

Once you go exec(), you never go back.

Therefore you need to create a new process to do the job. Then the parent (shell) can continue the execution.

http://joymepic.joyme.com/

Which is the best in exec() family in our case?

Member name	Using pathname	Using filename	Argument List	Argument Array	Original ENV	Provided ENV
execl()	YES		YES		YES	
execlp()		YES	YES		YES	
execle()	YES		YES			YES
execv()	YES			YES	YES	
execvp()		YES		YES	YES	
execve()	YES			YES		YES
Alphabet used in name		P	l	v		e

exec*() – arguments explained

Pathname VS Filename

/home/tywong/os/example.c

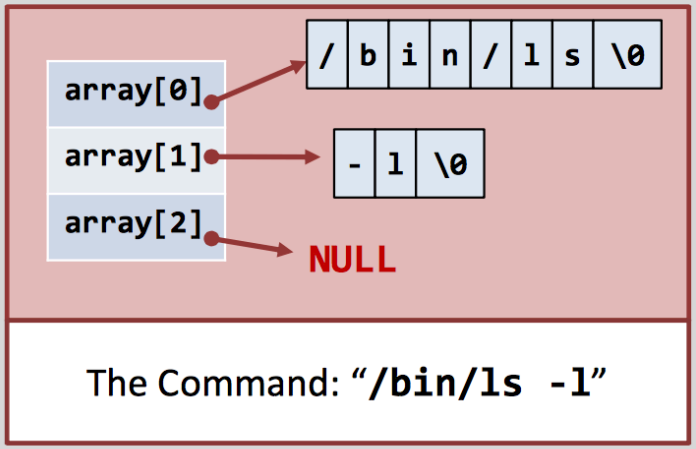
The entire string is a pathname.

The last part is a filename.

Argument list VS Argument array

```
execv("/bin/ls", array);
```

```
execl("/bin/ls",
"/bin/ls", "-l", NULL);
```



55

execvp() + setenv()

We use the combination of two system calls, `execvp()` and `setenv()`.

REQUIREMENT: (1) The shell allows file names (2)Search for the program

The sequence of the search path is determined by the environment variable **$PATH.**

What we do: (1) set the environment variable to our desired search path; (2) use execvp() to execute the external program.

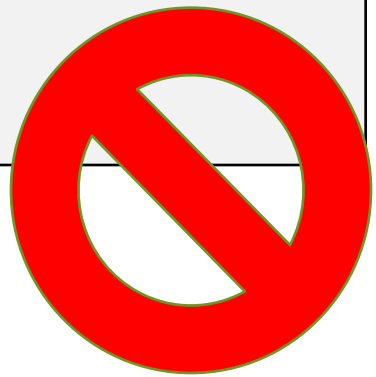
More on $PATH

<unistd.h>
<errno.h>

If we do not set the environment variable by ourselves, the program will follow the system's settings.

```
char *command1[] = {"shutdown",NULL};

printf("Running shutdown.. it is in /sbin :P \n\n");
execvp(*command1,command1);
if(errno == ENOENT)
        printf("No Command found...\n\n");
else
        printf("I dont know...\n");
```

/sbin/shutdown can be run!!!!!



Your program should disallow this

setenv()

<stdlib.h>

int setenv(const char *_name_, const char *_value_, int _overwrite_)

This function can change the environment variable in the system temporarily until the termination of the process.

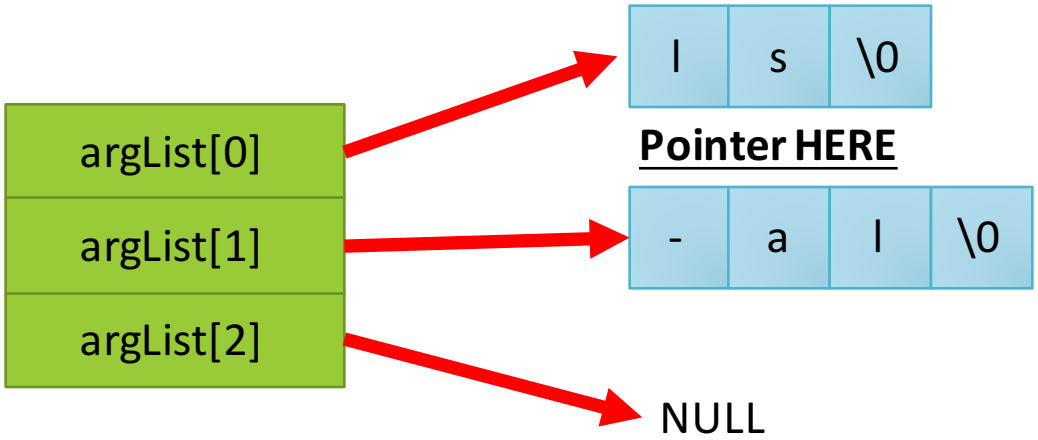
```
char *command1[] = {"shutdown",NULL};
printf("Running shutdown.. it is in /sbin :P \n\n");
setenv("PATH","/bin:/usr/bin:.",1);
execvp(*command1,command1);

if(errno == ENOENT)
        printf("No Command found...\n\n");
else
        printf("I dont know...\n");
return 0;
```

Now the shell only searches the specified path 

Create a dynamic argument array

To feed the arguments to execvp(), you need an argument array.

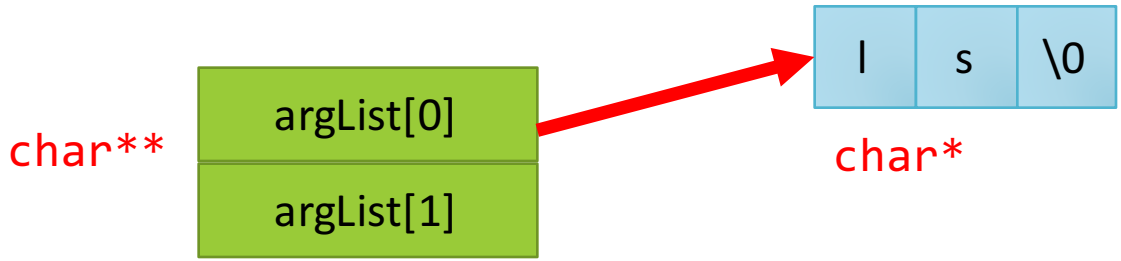
Remember in the first tutorial? The argument array is a **array of pointers.**



**argv

The **argv means the pointer needs to be dereferenced **twice**.

```
char **argList = (char**) malloc(sizeof(char*) * 3);
argList[0] = (char*)malloc(sizeof(char) * 10);
strcpy(argList[0],"ls");
argList[1] = (char*)malloc(sizeof(char) * 10);
strcpy(argList[1],"-al");
argList[2] = NULL;

execvp(*argList,argList);
```



fork() + exec()

<unistd.h>

We know that we need to fork a new process to use the system call exec().

```
#include <stdio.h>
#include <unistd.h> // Needed By fork(),sleep()
int main(int argc,char *argv[]){
        while(1)
        {
                printf("\nPress Enter to execute ls...");
                while(getchar() != '\n');
                if(!fork()) {
                        char *arglist[] = {"ls",NULL};
                        execvp(*arglist,arglist);
                }
                else{
                        sleep(1);
                }
        }
        return 0;
}
```

Is it okay? 👻

Zombies (wow…)

If the parent does not handle the termination of its child well, the child becomes a zombie 👻.

The child becomes zombie when the **parent is still alive** and **do not handle the event (signal).**

Zombie…

```
linux16:/uac/gds/hckam> ps aux | grep Z
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
hckam    15385  0.0  0.0      0     0 pts/1    Z+   21:07   0:00 [ls] <defunct>
hckam    15386  0.0  0.0      0     0 pts/1    Z+   21:07   0:00 [ls] <defunct>
hckam    15387  0.2  0.0      0     0 pts/1    Z+   21:07   0:00 [ls] <defunct>
hckam    15390  0.0  0.0      0     0 pts/1    Z+   21:07   0:00 [ls] <defunct>
hckam    15393  0.0  0.0   7840   848 pts/2    S+   21:07   0:00 grep Z
linux16:/uac/gds/hckam>
```

Mark will be deducted if you leave any zombies

How to handle the child properly

<unistd.h>
<sys/wait.h>
<sys/types.h>

The shell should wait until the child dies and handle the event.

The useful system call is **pid_t wait(int *_status_);**

```
while(1) {
        printf("\nPress Enter to execute ls...");
        while(getchar() != '\n');
        pid_t child_pid;
        if(!(child_pid = fork())) {
                char *arglist[] = {"ls",NULL};
                execvp(*arglist,arglist);
}
                else{
                        wait(NULL);
                }
        }
```

END

Happy Chinese New Year!

Remember to do the Phase 1!!