CSCI3150 TUTORIAL WEEK 5

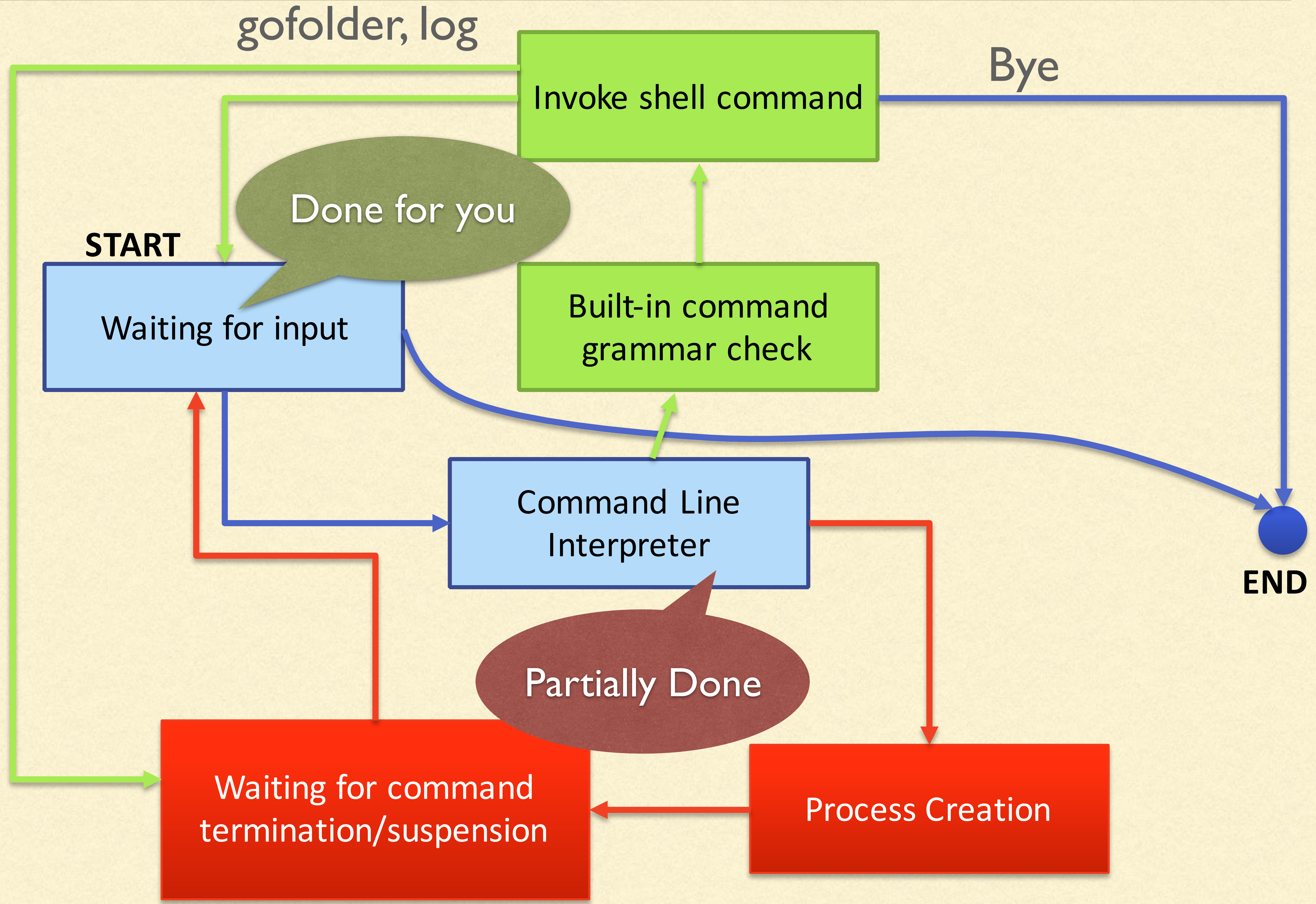
Calvin Kam (hckam@ CSE)

OUTLINE

- Introduction to Assignment 2
- Signals.
- Various functions you need.
- Command Chaining.
- Data Structure.

ASSIGNMENT 2 HAS BEEN RELEASED!

- Writing a Simple Shell
- Has following features:
  - Command Execution
  - Shell Commands
  - Signal Handling
  - Command Chaining using && and ||


gofolder, log
Invoke shell command
Bye
Done for you
START
Waiting for input
Built-in command grammar check
Command Line Interpreter
END
Partially Done
Waiting for command termination/suspension
Process Creation

SHELL COMMANDS

- Shell commands are the commands that you need to implement with your codes (not with exec*()).
- gofolder (~ cd)
- bye (~exit)
- log (~history)

THE WORKFLOW

- Ignore some signals
- Get the user input.
- Tokenize the input and store them.
- Check if it is a shell command. **Check grammar here.** Some of them has requirements.(e.g. Number of arguments). Execute if they are good to go. For other commands, use fork-exec-wait combination.
- Perform command chaining. Check the exit statuses and conditions.
- Once the child terminates, your shell prompts for user input.
- If user input "bye" or EOF (Ctrl-D), your program quits.

SIGNALS

- Signals are interrupts sent to the process.
- If custom signal handlers are not defined or not changed to ignore, default signal handler will be used.
- Eg: SIGSEGV(SEG Fault), SIGINT (Ctrl-C), **SIGTSTP** (Ctrl-Z), **SIGCHLD**, **SIGTERM**, **SIGKILL**, and etc.
- We use kill() to send out signals (not just kill the process!).
- BTW, **EOF** (Ctrl-D) is **NOT** signal

CUSTOM SIGNAL ROUTINES

- We can let the process behaves differently upon different signals.
- Can set them to <u>ignore</u> or even custom user-level handler.
- Of course, we cannot do anything on **SIGKILL** (unstoppable).

CUSTOM SIGNAL ROUTINES

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

int main(int argc,char *argv[])
{
    signal(SIGINT,SIG_IGN);
    printf("Put into while 1 loop..\n");
    while(1) { }
    printf("OK!\n");
    return 0;
}
```

SIG_IGN: Ignore

CUSTOM SIGNAL ROUTINES

```c
/* Signals/custom.c */
#include <stdio.h>
#include <signal.h>

void handler(int signal)
{
    printf("Signal %d Received.Kill me if you can\n",signal);
}

int main(int argc,char *argv[])
{
    signal(SIGINT,handler);
    printf("Put into while 1 loop..\n");
    while(1) { }
    printf("OK!\n");
    return 0;
}
```

GET CURRENT PATH

- In the prompt, it shows the current working directory.
- We can use a getcwd() to get it easily.

```
#include <unistd.h>
char *getcwd(char *buf, size_t size);
```

GETCWD()

```
#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 DIRECTORY

- To change the working directory, we can use the following function.

- “gofolder” in your assignment.

```c
#include <unistd.h>
int chdir(const char *path);
```

CHDIR()

```c
/* Shell/chdir.c */
#include <stdio.h>
#include <unistd.h>
#include <limits.h>
#include <errno.h>
#include <string.h>
int main(int argc,char *argv[]) {
    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);
        input[strlen(input)-1] = '\0';
        if(chdir(input) != -1) {
            getcwd(buf,PATH_MAX+1);
            printf("Now it is %s\n",buf);       }
        else {
            printf("Cannot Change Directory\n"); }
    }
return 0;
}
```

Error Checking

CHANGE ENVIRONMENT VARIABLE

- In order to setup a searching sequence for shell, we need to change the $PATH variable.
- You can either provide a new set of environment variables in some of exec*() family members or using this function:

```
#include <stdlib.h>
int setenv(const char *name, const char *value, int overwrite);
```

SETENV()

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>

int main(int argc,char *argv[])
{
    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;
}
```

Need To Overwrite

COMMAND CHAINING

- In this assignment, you are required to chain commands by AND (&&) and OR (||).
- These are very useful in shell script programming.
- A (Operator) B.
  - Execution of B depends on the exit status of A.
  - Runs successfully = Exit Normally with Exit Status 0.
- In assignment we won’t test you for chaining many commands, but of course try take the challenges ;P

AND &&

- A && B
- If A runs <u>successfully</u>, then B will run.
- If A fails, then B will NOT run.
- Usage:
  - Doing a series of jobs of which the subsequent job only runs if the previous one runs successfully.
  - mkdir abc && cd abc

OR (||)

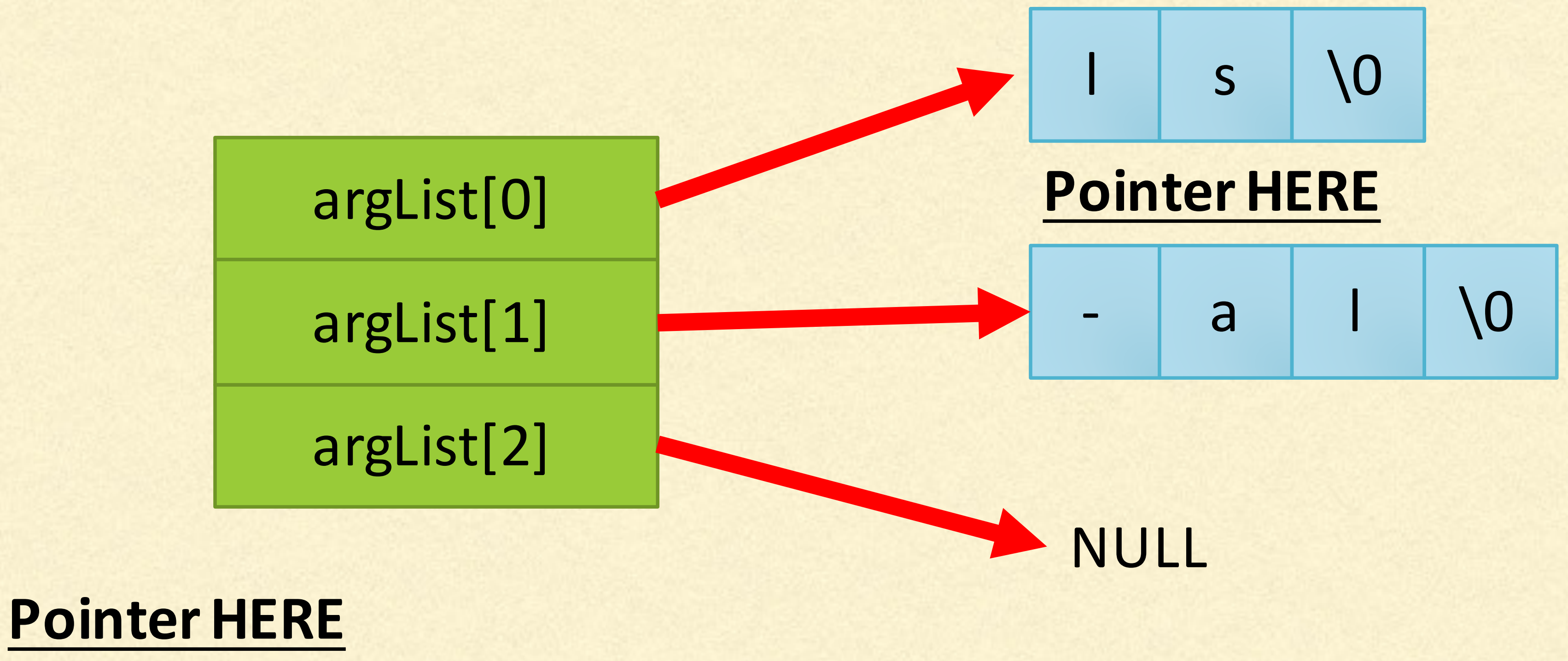
- A || B
- If A runs <u>successfully</u>, B will NOT run.
- If A <u>fails</u>, then B will run.
- Usage:
  - Error Reporting.
  - rm abc && echo “Error”.

DATA STRUCTURE

- Remember your data structure class :P
- It is good to store your commands in a manageable data structure.
- Multi-dimensional arrays
- Linked List, vector
- and etc….

ARGUMENT ARRAY

- In some exec*() members, you need to provide an argument array.
- Actually it is an array of pointers.



ARGUMENT ARRAY

- As we have to access two pointers in order to get to the string, dereferencing twice (malloc() two times) are needed.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>

int main(int argc,char *argv[])
{
    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);
   return 0;
}
```

GOOD LUCK 😎

- Individual Work.
- Prof. Lo 's Early Bird Policy
- Early Bird Submission Deadline: 13 OCT 2016 11:59 **AM**
- Normal Submission Deadline: 20 OCT 2016 11:59 **AM**