Design Document - HW1

Student ID: B112040003
Department and Class: Computer Science 115
Name: 張景旭

1. Overview

Identifying processes

- **the goal:**

> to recognize the correct process, and execute the processs

fork

- **goal:**

> to clone the calling process, creating an exact copy.Return -1 for errors, O to the new process,and the process ID of the new process to the old process.

execvp

```
int execvp(const char *file, char *const argv[]);
```

- **goal:**

> to execute file to replace current process, and pass the arguments argv[]

waitpid/exit

- **goal:**

> wait until the child process call exit() when the child process is done

dup2

- **goal:**

  > to modify the file discripter, changing to stdin(0) and stdout(1) to redirect input/output file and pipes

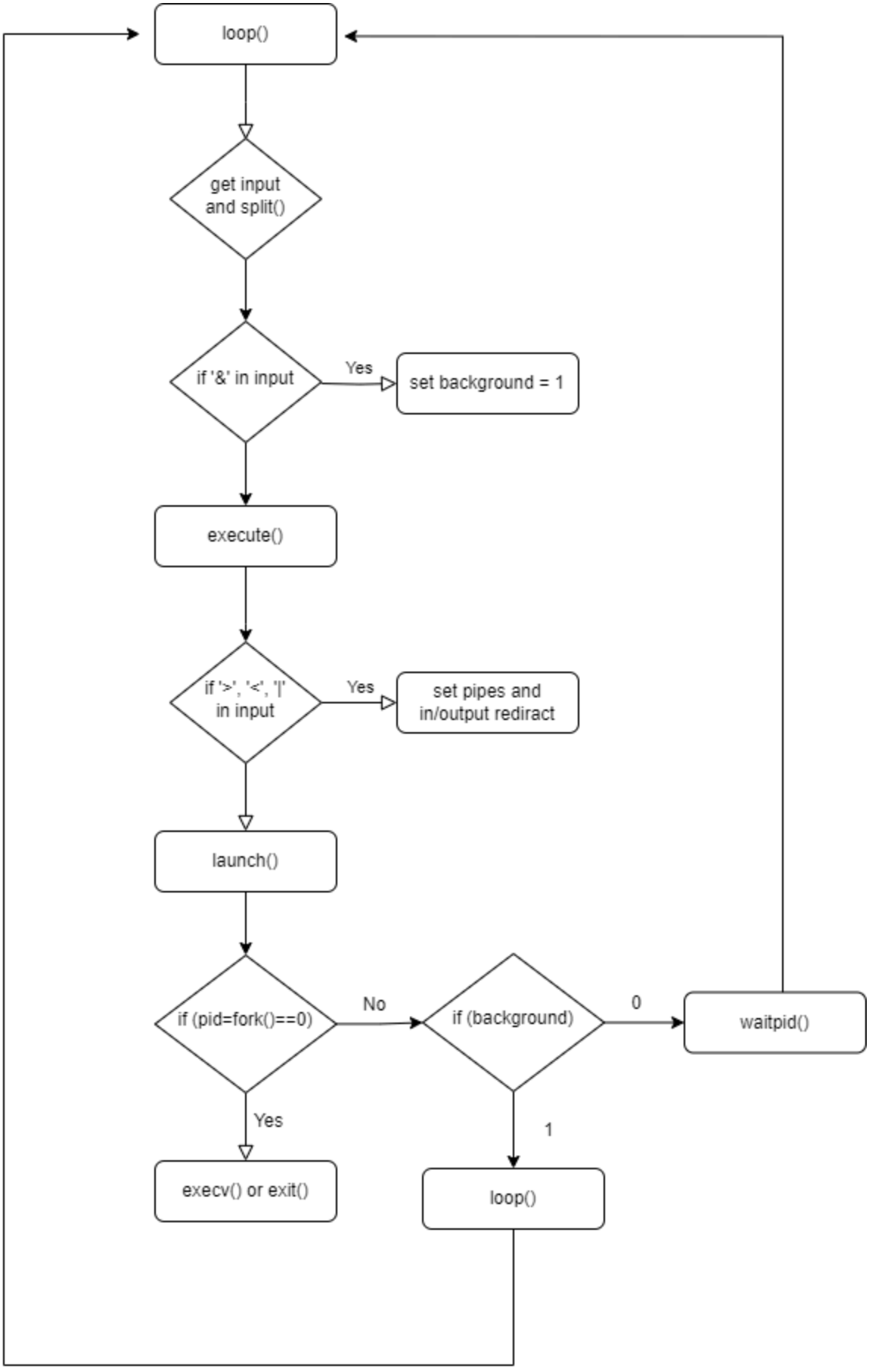
pipe

```
int pipe(int pipefd[2]);
```

- **goal:**

  > to let one process output(piepfd[1]) to be another process input(pipefd[0])

How the different parts of the design interact together



Major algorithms

1. split the input string to char** by spaces
2. check if the processes running in background(&)
3. check for pipe(|) or input/output redirect(>,<)
4. create pipe and duplicate the file discripter
5. call fork() and execvp() to execute the processes
6. wait the processes be done if it is not running in background

Major data structures

1. vector
2. string

2. In-depth analysis and implementation

The functions to be implemented

- **split()**

> 1. use getline() to split the input string by spaces
> 2. push the result back to the vector＜string＞
> 3. turn vector＜string＞ to char** and return

  - pseudo code

```
Function split(str, delimiter):
    result = empty vector<string>
    ss = create stringstream with str
    tok = empty string
    while get line from ss with delimiter into tok:
        add tok to result
    if result is empty:
        create array of char pointers with size 1
        set first element to nullptr
        return array
    create array of char pointers with size result.size() + 1
    for each string in result:
        create char array with length string.length() + 1
        copy string into char array
        add char array to array of char pointers
    set last element of array to nullptr
    return array
```

- **builtin_cd()**

> 1. if input argument is empty, change to directory to home
> 2. change the directory by arguemt

- **bulitin_exit()**

> 1. return 0 to exit the program

- **loop()**

> 1. identifying processes
> 2. check if the processes running in background
> 3. return the arguments to the execute()

```
Function loop():
str = empty string
tokens = empty array of char pointers
delimiter = ' '
background = false
login_name = NULL
pwd_dirname = NULL

login_name = get environment variable "LOGNAME"
pwd_dirname = get current working directory
do:
    display login_name, pwd_dirname, and shell prompt
    read input into str from standard input
    if str is not empty and last character is '&':
        if background is false:
            remove last character from str (remove '&')
            set background to true
    tokens = split str using delimiter
    if str is empty:
        continue to next iteration
    status=execute command specified by tokens,considering background ex
    for each token in tokens:
        free memory allocated for token
    free memory allocated for tokens
    if background is true:
        set background to false
    if status indicates exit command:
        exit loop
while true
```

- **execute()**

> 1. check if pipe(|) or input/output redirect(>,<)
> 2. call launnch

```
Function execute(args, background = false):
    if args is nullptr:
        return 1
    if first element of args is NULL:
        return 1
    commands = empty vector of vector of strings
    command = empty vector of strings
    input_redirect = false
    output_redirect = false
    input_file = empty string
    output_file = empty string
    for each argument in args:
        if argument is "<":
            set input_redirect to true
            set input_file to next argument (input file)
            skip next argument
        else if argument is ">":
            set output_redirect to true
            set output_file to next argument (output file)
            skip next argument
        else if argument is "|":
            if command is not empty:
                add command to commands
                clear command
        else:
            add argument to command
    if command is not empty:
        add command to commands

    return launch commands considering background,
    input redirection, output redirection, input file, and output file
```

- **launch()**

> 1. create pipe and duplicate the file diractor
> 2. call fork() to execute the process
> 3. wait the processes be done if it is not running in background

Corner cases that need to be handled (list all possible error conditions returned from each system call)

- fork error : if pid = fork() == -1
- execvp error : if execvp() return -1
- open error : if open() return - 1
- pipe error : if pipe() return -1
- cd error : if chdir() return -1

Test Plan

Possible Inputs	Expected Output	Acual Output
ls	main main.cpp makefile	main main.cpp makefile
ls &	$main main.cpp makefile	$main main.cpp makefile
ls -l > foo	(a file foo with context of ls -l)	(a file foo with context of ls -l)
ls | grep main	main main.cpp	main main.cpp
|	-sh: Syntax error: "|" unexpected	Segmentation fault
>	-sh: Syntax error: newline unexpected	Segmentation fault
<	-sh: Syntax error: newline unexpected	Segmentation fault
&	-sh: Syntax error: "&" unexpected	
a	a: not found	a: No such file or directory
echo hi |	> (wait for input)	hi

3. Risk

- Risk Identification

  > if input be "|", ">" or "<", the program will crash with Segmentation fault (core dumped)

- **Risk Assessment**

  > The occurrence of a system coredump may result in insufficient disk space, leading to errors in the execution of other systems or applications.

- **Risk Treatment**

  > Implement input validation to detect whether it is a single special character, reducing system coredumps.