**Operating Systems-Theory**

**Assignment 1**

**Muhammad Abdullah**

**L1f21bscs0379**

**E5**

**Problem Statement**

Write a C program for UNIX to implement your own shell that resembles BASH. (5 x 4 Marks)

* The shell should be capable of running programs in both the foreground and background.

|  |
| --- |
| Example |
| > ./a.out  > ./a.out & |

**solution**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

#define MAX\_INPUT\_SIZE 1024

int main()

{

char input[MAX\_INPUT\_SIZE];

while (1)

{

printf("> ");

fflush(stdout);

fgets(input, sizeof(input), stdin);

if (input[0] == '\n')

{

continue;

}

input[strlen(input) - 1] = '\0';

if (strcmp(input, "exit") == 0)

{

break;

}

int background = 0;

{

background = 1;

input[strlen(input) - 1] = '\0';

}

pid\_t pid = fork();

if (pid == 0)

{

system(input);

exit(EXIT\_SUCCESS);

}

else if (pid < 0)

{

perror("Fork failed");

exit(EXIT\_FAILURE);

}

if (!background)

{

waitpid(pid, NULL, 0);

}

}

return 0;

}

* The shell should support input and output redirection.

|  |
| --- |
| Example |
| > ls > output.txt  > ls -l / > output.txt  >sort < input.txt > output.txt |

**Solution:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

#include <fcntl.h>

#define MAX\_INPUT\_SIZE 1024

int main()

{

char input[MAX\_INPUT\_SIZE];

while (1)

{

printf("> ");

fflush(stdout);

fgets(input, sizeof(input), stdin);

if (input[0] == '\n')

{

continue;

}

input[strlen(input) - 1] = '\0';

if (strcmp(input, "exit") == 0)

{

break;

}

int background = 0;

if (input[strlen(input) - 1] == '&')

{

background = 1;

input[strlen(input) - 1] = '\0';

}

char \*input\_redirect = strstr(input, "<");

char \*output\_redirect = strstr(input, ">");

if (input\_redirect != NULL)

{

\*input\_redirect = '\0';

input\_redirect += 2;

int fd\_input = open(input\_redirect, O\_RDONLY);

if (fd\_input == -1)

{

perror("Error opening input file");

continue;

}

dup2(fd\_input, STDIN\_FILENO);

close(fd\_input);

}

if (output\_redirect != NULL)

{

\*output\_redirect = '\0';

output\_redirect += 2;

int fd\_output = open(output\_redirect, O\_WRONLY | O\_CREAT | O\_TRUNC, 0666);

if (fd\_output == -1)

{

perror("Error opening output file");

continue;

}

dup2(fd\_output, STDOUT\_FILENO);

close(fd\_output);

}

pid\_t pid = fork();

if (pid == 0)

{

// Child process

system(input);

exit(EXIT\_SUCCESS);

}

else if (pid < 0)

{

perror("Fork failed");

exit(EXIT\_FAILURE);

}

if (!background)

{

waitpid(pid, NULL, 0);

}

}

return 0;

}

* The shell should support piping between processes.

|  |
| --- |
| Example |
| > ls | sort -R | grep “bin” |

**Solution:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

#include <fcntl.h>

#define MAX\_INPUT\_SIZE 1024

#define MAX\_COMMANDS 10

int main()

{

char input[MAX\_INPUT\_SIZE];

char \*commands[MAX\_COMMANDS];

int command\_count = 0;

while (1)

{

printf("> ");

fflush(stdout);

fgets(input, sizeof(input), stdin);

if (input[0] == '\n')

{

continue;

}

input[strlen(input) - 1] = '\0';

if (strcmp(input, "exit") == 0)

{

break;

}

int background = 0;

if (input[strlen(input) - 1] == '&')

{

background = 1;

input[strlen(input) - 1] = '\0';

}

char \*token = strtok(input, "|");

int cmd\_index = 0;

while (token != NULL)

{

commands[cmd\_index++] = token;

token = strtok(NULL, "|");

}

commands[cmd\_index] = NULL;

int pipes[2];

int input\_fd = 0;

for (int i = 0; commands[i] != NULL; i++)

{

pipe(pipes);

pid\_t pid = fork();

if (pid == 0)

{

dup2(input\_fd, STDIN\_FILENO);

if (commands[i + 1] != NULL)

{

dup2(pipes[1], STDOUT\_FILENO);

}

close(pipes[0]); // Close unused pipe end

execlp(commands[i], commands[i], NULL);

perror("Execution failed");

exit(EXIT\_FAILURE);

}

else if (pid < 0)

{

perror("Fork failed");

exit(EXIT\_FAILURE);

}

close(pipes[1]); // Close write end of the pipe

input\_fd = pipes[0];

}

if (!background)

{

for (int i = 0; i < command\_count; i++)

{

wait(NULL);

}

}

command\_count++;

}

return 0;

}

* The shell should support a combination of piping between processes and redirections.

|  |
| --- |
| Example |
| > sort -R <input.txt | grep “bin” > output.txt |

**Solution:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

#include <fcntl.h>

#define MAX\_INPUT\_SIZE 1024

#define MAX\_COMMANDS 10

int main()

{

char input[MAX\_INPUT\_SIZE];

char \*commands[MAX\_COMMANDS];

int command\_count = 0;

while (1)

{

printf("> ");

fflush(stdout);

fgets(input, sizeof(input), stdin);

if (input[0] == '\n')

{

continue;

}

input[strlen(input) - 1] = '\0';

if (strcmp(input, "exit") == 0)

{

break;

}

int background = 0;

if (input[strlen(input) - 1] == '&')

{

background = 1;

input[strlen(input) - 1] = '\0';

}

char \*token = strtok(input, "|");

int cmd\_index = 0;

while (token != NULL)

{

commands[cmd\_index++] = token;

token = strtok(NULL, "|");

}

commands[cmd\_index] = NULL;

int pipes[2];

int input\_fd = 0;

for (int i = 0; commands[i] != NULL; i++)

{

char \*output\_redirect = strstr(commands[i], ">");

char \*input\_redirect = strstr(commands[i], "<");

if (output\_redirect != NULL)

{

\*output\_redirect = '\0';

output\_redirect += 2;

int fd\_output = open(output\_redirect, O\_WRONLY | O\_CREAT | O\_TRUNC, 0666);

if (fd\_output == -1)

{

perror("Error opening output file");

exit(EXIT\_FAILURE);

}

dup2(fd\_output, STDOUT\_FILENO);

close(fd\_output);

}

else if (input\_redirect != NULL)

{

\*input\_redirect = '\0';

input\_redirect += 2;

int fd\_input = open(input\_redirect, O\_RDONLY);

if (fd\_input == -1)

{

perror("Error opening input file");

exit(EXIT\_FAILURE);

}

dup2(fd\_input, STDIN\_FILENO);

close(fd\_input);

}

pipe(pipes);

pid\_t pid = fork();

if (pid == 0)

{

dup2(input\_fd, STDIN\_FILENO);

if (commands[i + 1] != NULL)

{

dup2(pipes[1], STDOUT\_FILENO);

}

close(pipes[0]);

execlp(commands[i], commands[i], NULL);

perror("Execution failed");

exit(EXIT\_FAILURE);

}

else if (pid < 0)

{

perror("Fork failed");

exit(EXIT\_FAILURE);

}

close(pipes[1]);

input\_fd = pipes[0];

}

if (!background)

{

for (int i = 0; i < command\_count; i++)

{

wait(NULL);

}

}

command\_count++;

}

return 0;

}