Slip 25

Program 1: Child Process with Fork and Execve

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

int main() {
   pid_t pid;
   char *args[] = {"/bin/ls", "-l", NULL};  // Arguments for execve

   pid = fork();

   if (pid == 0) {
      // Child process
      printf("Child Process: Executing 'ls -l' using execve()\n");
      execve("/bin/ls", args, NULL);  // Execute ls command in child process
      printf("This line will not be printed if execve is successful.\n");
   }
   else if (pid > 0) {
      // Parent process
      wait(NULL);  // Wait for the child process to complete
      printf("Parent Process: Child completed\n");
   }
   else {
      printf("Fork failed!\n");
   }

   return 0;
}
```

Program 2: Shell with `search` Command

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

void search_file(char *option, char *filename, char *pattern) {
   FILE *file = fopen(filename, "r");
```

```
    if (file == NULL) {
        printf("File %s not found.\n", filename);
        return;
    }

    char line[256];
    int count = 0, line_num = 0;

    while (fgets(line, sizeof(line), file)) {
        line_num++;
        if (strstr(line, pattern) != NULL) {
            if (strcmp(option, "a") == 0) {
                printf("Line %d: %s", line_num, line);
            }
            count++;
        }
    }

    if (strcmp(option, "c") == 0) {
        printf("Pattern '%s' occurred %d times in file %s.\n", pattern, count, filename);
    }

    fclose(file);
}

int main() {
    char command[100], *args[10];
    while (1) {
        printf("\nmyshell$ ");
        fgets(command, 100, stdin);
        command[strlen(command) - 1] = '\0';  // Remove newline

        char *token = strtok(command, " ");
        int i = 0;
        while (token != NULL) {
            args[i++] = token;
            token = strtok(NULL, " ");
        }
        args[i] = NULL;
```

```
        if (strcmp(args[0], "search") == 0) {
            search_file(args[1], args[2], args[3]);
        } else if (strcmp(args[0], "exit") == 0) {
            exit(0);
        } else {
            int pid = fork();
            if (pid == 0) {
                execvp(args[0], args);
                exit(0);
            } else {
                wait(NULL);
            }
        }
    }
    return 0;
}
```