Process management

1. fork - Link 1, Link 2
    - `pid_t fork(void);`
    - fork() creates a new process by duplicating the calling process. It returns a negative value when the function fails to create a child process. On successful duplication of a process, the PID of the child process is returned in the parent, and 0 is returned in the child process.
2. wait(NULL)
    - wait(NULL) will block the parent process until any of its children has finished. If the child terminates before the parent process reaches wait(NULL), then the child process turns to a zombie process until its parent waits on it and it is released from memory.
3. execlp
    - `int execlp(const char *file, const char *arg, ...)`
    - The file argument is the path name of an executable file to be executed. arg is the string we want to appear as argv[0] in the executable. By convention, argv[0] is just the executable file name; normally, it's set to the same as the file. The ... are now the additional arguments to give to the executable.
4. fopen - Link 1
    - `FILE *fopen(const char *pathname, const char *mode)`
    - The `fopen()` function opens the file whose name is the string pointed to by pathname and associates a stream with it.
5. fclose - Link 1
    - `int fclose(FILE *stream)`
    - The `fclose()` function flushes the stream pointed to by the stream (writing any buffered output data using `fflush()`) and closes the underlying file descriptor.
6. fgetc - Link 1
    - `int fgetc(FILE *pointer)`
    - fgetc() is used to obtain input from a file, a single character at a time. This function returns the ASCII code of the character read by the function. `pointer` is a pointer to a FILE object that identifies the stream on which the operation is to be performed.

Problem 0

Write a C program that creates a child process to run the `ls` command and ensures that the parent process waits for the child to finish before exiting.

Solution

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

int main(int argv, char *argc[]) {
  // Create a child process
  int child_pid = fork();
  if (child_pid == 0) {
     // In the child process, execute a bash command
     execlp("ls", "ls", NULL);
  } else {
     // Wait for the child process to terminate
     wait(NULL);
     printf("The child process has finished execution.\n");
  }
}
```