

Practice #4

1.

Main.c

+

43bywbkjp

```
1 #include <sys/types.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 int main()
5 { pid_t pid; /* fork a child process */
6   pid = fork();
7   if (pid < 0) { /* error occurred */
8     fprintf(stderr, "Fork Failed");
9     return 1;
10  }
11  else if (pid == 0) { /* child process */
12    execlp("/bin/ls", "ls", NULL);
13  }
14  else { /* parent process */
15    /* parent will wait for the child to complete */
16    wait(NULL);
17    printf("Child Complete");
18  }
19  return 0;
20 }
21
```

STDIR

Input for the program (Optional)

Output:

Main

Main.c

Child Complete

2.

[illegible]

3.

Main.c

+

43bywbkjp

AI NEW C RUN

```
1 #include <sys/types.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 int main()
5 {
6     pid_t pid, pid1;
7     /* fork a child process */
8     pid = fork();
9     if (pid < 0) { /* error occurred */
10         fprintf(stderr, "Fork Failed");
11         return 1;
12     }
13     else if (pid == 0) { /* child process */
14         pid1 = getpid();
15         printf("child: pid = %d",pid); /* A */
16         printf("child: pid1 = %d",pid1); /* B */
17     }
18     else { /* parent process */
19         pid1 = getpid();
20         printf("parent: pid = %d",pid); /* C */
21         printf("parent: pid1 = %d",pid1); /* D */
22         wait(NULL);
23     }
24     return 0;
25 }
26
```

STDIN

Input for the program (Optional)

Output:

child: pid = 0child: pid1 = 6413parent: pid = 6413parent: pid1

4.

Main.c

+

43bywbkjp

AI NEW C RUN

```
1 #include <sys/types.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 int value = 5;
5 int main()
6 {
7     pid_t pid;
8     pid = fork();
9     if (pid == 0) { /* child process */
10         value += 15;
11         return 0;
12     }
13     else if (pid > 0) { /* parent process */
14         wait(NULL);
15         printf("PARENT: value = %d",value); /* LINE A */
16         return 0;
17     }
18 }
19
```

STDIN

Input for the program (Optional)

Output:

PARENT: value = 5

5.

Main.c

+

43bywbkjp

AI NEW C RUN

```
1 #include <sys/types.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 #define SIZE 5
5 int nums[SIZE] = {0,1,2,3,4};
6 int main()
7 { int i;
8   pid_t pid;
9   pid = fork();
10  if (pid == 0) {
11      for (i = 0; i < SIZE; i++) { nums[i] *= -i;
12          printf("CHILD: %d ",nums[i]); /* LINE X */ }
13      else if (pid > 0) { wait(NULL);
14          for (i = 0; i < SIZE; i++)
15              printf("PARENT: %d ",nums[i]); /* LINE Y */
16      }
17      return 0;
18  }
19
```

STDIN

Input for the program (Optional)

Output:

Main.c: In function 'main':
Main.c:8:1: error: unknown type name 'pid'; did you mean 'pid_t'
8 | pid_t pid;
| ~~~
| pid_t
Main.c:8:7: error: expected '=', ',', ';', 'asm' or '__attribu
8 | pid_t pid;
| ~~~
Main.c:9:1: error: 'pid' undeclared (first use in this functio
9 | pid = fork();
| ~~~
Main.c:9:1: note: each undeclared identifier is reported only
Main.c:13:21: warning: implicit declaration of function 'wait'
13 | else if (pid > 0) { wait(NULL);
| ~~~

```
Main.c 43bywbkpi
1 #include <sys/types.h>
2 #include <sys/wait.h>
3 #include <stdio.h>
4 #include <unistd.h>
5
6 #define SIZE 5
7 int nums[SIZE] = {0, 1, 2, 3, 4};
8
9 int main() {
10     int i;
11     pid_t pid;
12
13     pid = fork();
14     if (pid == 0) {
15         for (i = 0; i < SIZE; i++) {
16             nums[i] *= -i;
17             printf("CHILD: %d ", nums[i]); // LINE X
18         }
19     } else if (pid > 0) {
20         wait(NULL);
21         for (i = 0; i < SIZE; i++) {
22             printf("PARENT: %d ", nums[i]); // LINE Y
23         }
24     }
25     return 0;
26 }
27
```

STDIN

Input for the program (Optional)

Output:

CHILD: 0 CHILD: -1 CHILD: -4 CHILD: -9 CHILD: -16 PARENT: 0

Explanation of the code

This code demonstrates creating a child process using `fork()` and working with a shared array `nums`.

Code breakdown:

1. Including libraries

- `sys/types.h` and `unistd.h` — for system calls (`fork()`).
- `sys/wait.h` — for using `wait()`, so the parent process waits for the child.
- `stdio.h` — for input-output functions (`printf()`).

2. Declaring variables

- `nums[SIZE] = {0, 1, 2, 3, 4};` — an array used by both the parent and child process.
- `pid_t pid;` — a variable to store the process ID.

3. Creating a new process

- `pid = fork();`
 - If `pid == 0`, it means the child process is running.
 - If `pid > 0`, it means the parent process is running.

4. Processing in the child process

- Loops through `nums`, multiplies each element by `-i`.
- Prints values to the console (CHILD: ...).

5. Processing in the parent process

- The parent waits for the child process to finish (`wait(NULL);`).
- After the child process finishes, it prints the original values (PARENT: ...).

Changes that were made:

1. Fixed the pid declaration error

- Before: pid t pid;
- Now: pid_t pid;

2. Added #include <sys/wait.h>

- Needed for wait().