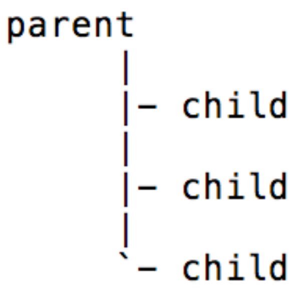
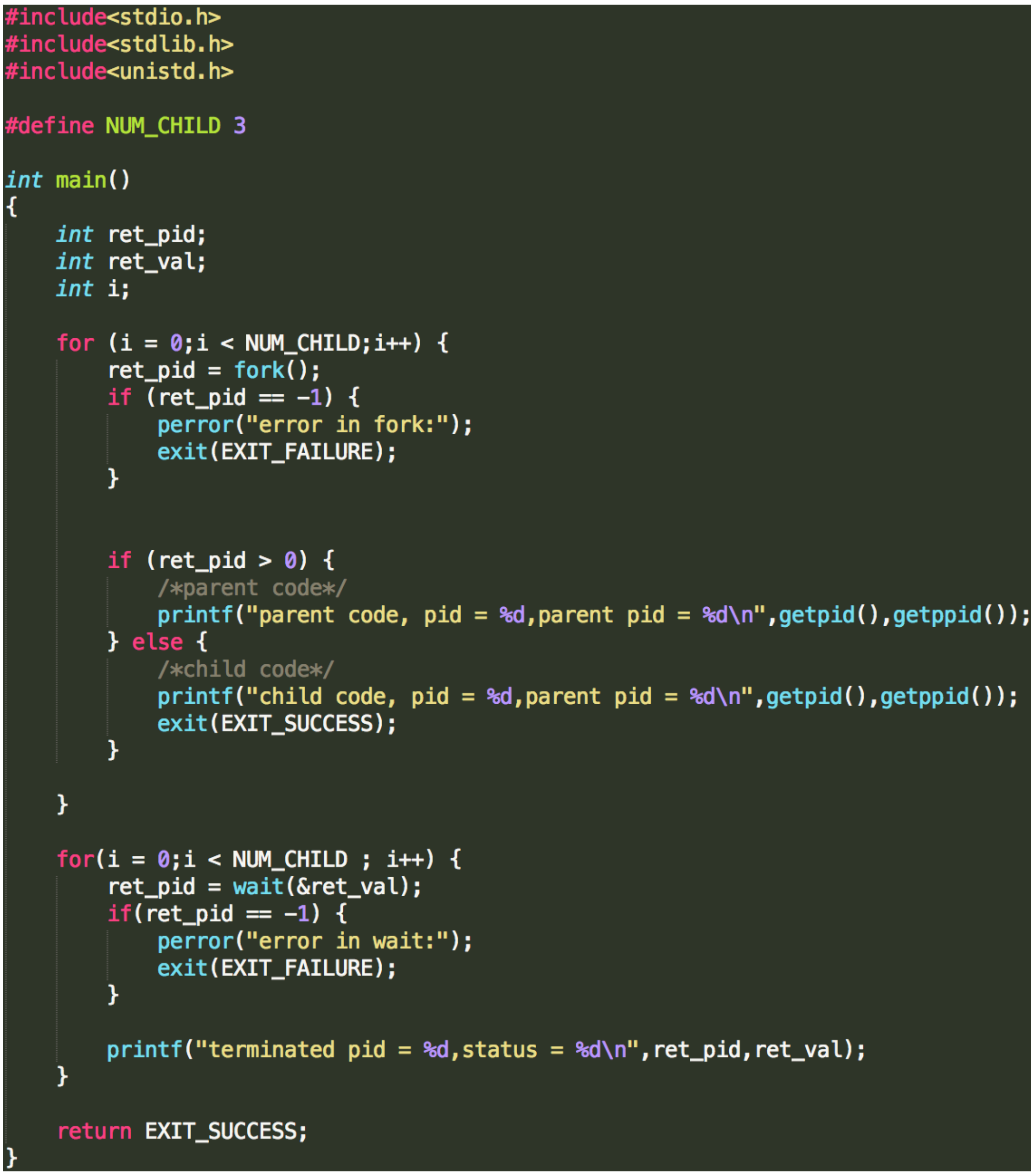
1.**请用 PV 解决“过独木桥”问题：**同一方向行人可连续过桥，当某一方向有人过桥时，另一方向的行人必须等待；当某一方向无人过桥时，另一方向的行人可以过桥。

```
Semaphore right=1, left=1, brige_mutex=1;
int right_count=0, left_count=0;
process 左() {
    while(true) {
        P(left);
        left_count++;
        if (left_count==1) P(brige_mutex);
        V(left);
        过独木桥;
        P(left);
        left_count--;
        if(left_count==0) V(brige_mutex);
        V(left);
    }
}
process 右() {
    while(true) {
        P(right);
        right_count++;
        if (right_count==1) P(brige_mutex);
        V(right);
        过独木桥;
        P(right);
        right_count--;
        if(right_count==0) V(brige_mutex);
        V(right);
    }
}
```

3. 请用 **fork()**系统调用创建如下关系的父子进程

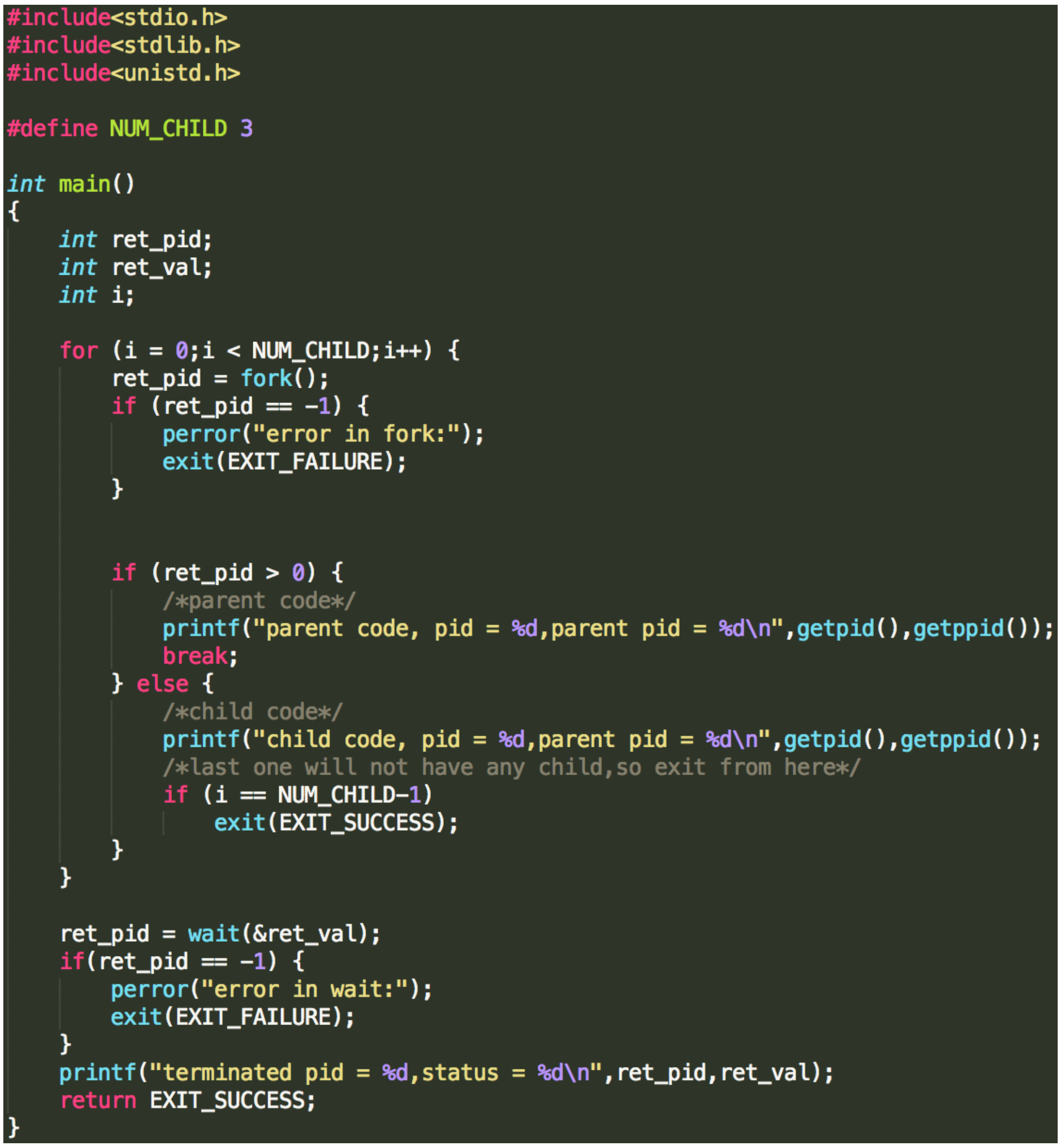


```
parent
      |
      |- child
      |
      |- child
      |
      `- child
```



```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

#define NUM_CHILD 3

int main()
{
    int ret_pid;
    int ret_val;
    int i;

    for (i = 0;i < NUM_CHILD;i++) {
        ret_pid = fork();
        if (ret_pid == -1) {
            perror("error in fork:");
            exit(EXIT_FAILURE);
        }


        if (ret_pid > 0) {
            /*parent code*/
            printf("parent code, pid = %d,parent pid = %d\n",getpid(),getppid());
        } else {
            /*child code*/
            printf("child code, pid = %d,parent pid = %d\n",getpid(),getppid());
            exit(EXIT_SUCCESS);
        }

    }

    for(i = 0;i < NUM_CHILD ; i++) {
        ret_pid = wait(&ret_val);
        if(ret_pid == -1) {
            perror("error in wait:");
            exit(EXIT_FAILURE);
        }

        printf("terminated pid = %d,status = %d\n",ret_pid,ret_val);
    }

    return EXIT_SUCCESS;
}
```

```
parent
        |
        child
              |
              child
                    |
                    child
```



```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

#define NUM_CHILD 3

int main()
{
    int ret_pid;
    int ret_val;
    int i;

    for (i = 0;i < NUM_CHILD;i++) {
        ret_pid = fork();
        if (ret_pid == -1) {
            perror("error in fork:");
            exit(EXIT_FAILURE);
        }


        if (ret_pid > 0) {
            /*parent code*/
            printf("parent code, pid = %d,parent pid = %d\n",getpid(),getppid());
            break;
        } else {
            /*child code*/
            printf("child code, pid = %d,parent pid = %d\n",getpid(),getppid());
            /*last one will not have any child,so exit from here*/
            if (i == NUM_CHILD-1)
                exit(EXIT_SUCCESS);
        }
    }

    ret_pid = wait(&ret_val);
    if(ret_pid == -1) {
        perror("error in wait:");
        exit(EXIT_FAILURE);
    }
    printf("terminated pid = %d,status = %d\n",ret_pid,ret_val);
    return EXIT_SUCCESS;
}
```