Code examples for shell lab (Supplementary slides)

Yogesh Virkar

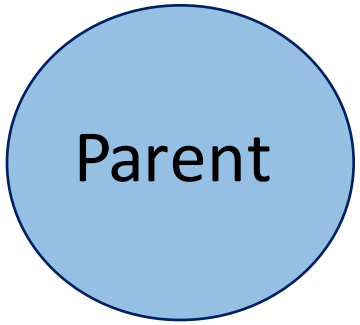
Department of Computer Science,

University of Colorado, Boulder.

Parent's process ID: 9812

children-do-not-wait.c

```
while (birthed++ < kids ) {
    fprintf(stderr, "%d: fork child\n", getpid());
    if ((pid = Fork()) == 0) {
      fprintf(stderr, "%d: I am the child\n", getpid());
      exit(0);
    }
    fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
  }
```



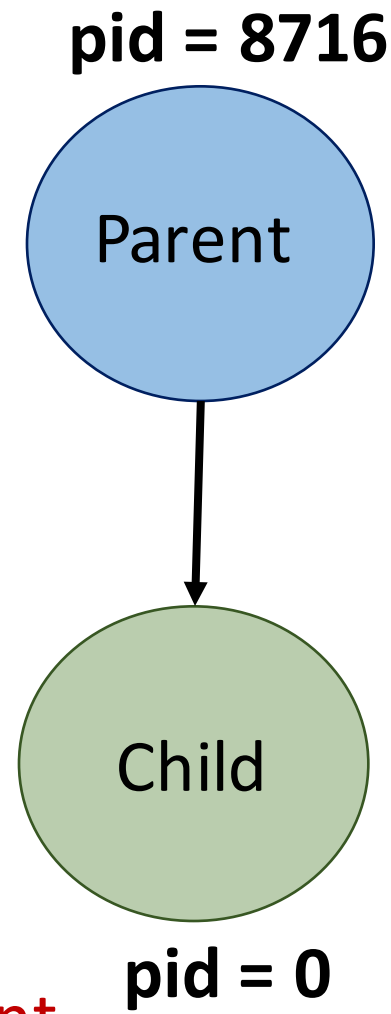
children-do-not-wait.c (2)

Parent's process ID: 9812
Child's process ID: 8716

```
while (birthed++ < kids ) {
    fprintf(stderr, "%d: fork child\n", getpid());
    if ((pid = Fork()) == 0) {
        fprintf(stderr, "%d: I am the child\n", getpid());
        exit(0);
    }
    fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
}
```

- Parent
- Child

Parent executes fork()
fork() returns twice--- once for parent, once for child



children-do-not-wait.c (3)

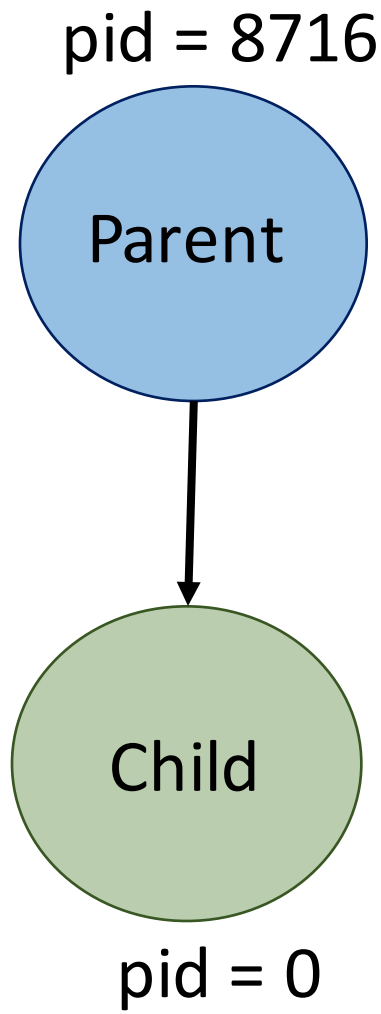
fprintfs

3883: Child is 3884

3884: I am the child

```
while (birthed++ < kids ) {
    fprintf(stderr, "%d: fork child\n", getpid());
    if ((pid = Fork()) == 0) {
        fprintf(stderr, "%d: I am the child\n", getpid());
        exit(0);
    }
    fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
}
```

- Parent
- Child



children-do-not-wait.c (4)

fprintfs

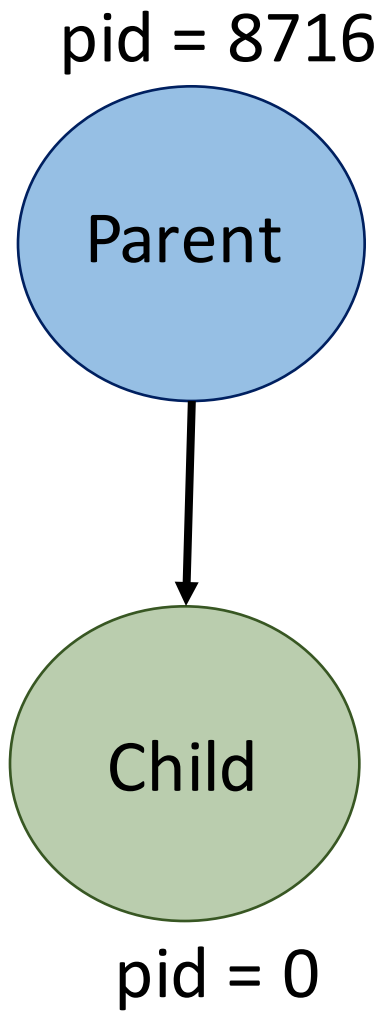
3883: Child is 3884

3884: I am the child

Ordering of printfs is not guaranteed

```
while (birthed++ < kids ) {
    fprintf(stderr, "%d: fork child\n", getpid());
    if ((pid = Fork()) == 0) {
        fprintf(stderr, "%d: I am the child\n", getpid());
        exit(0);
    }
    fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
}
```

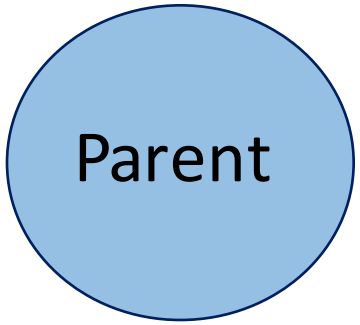
- Parent
- Child



children-simple-wait.c

Parent's process ID: 9812

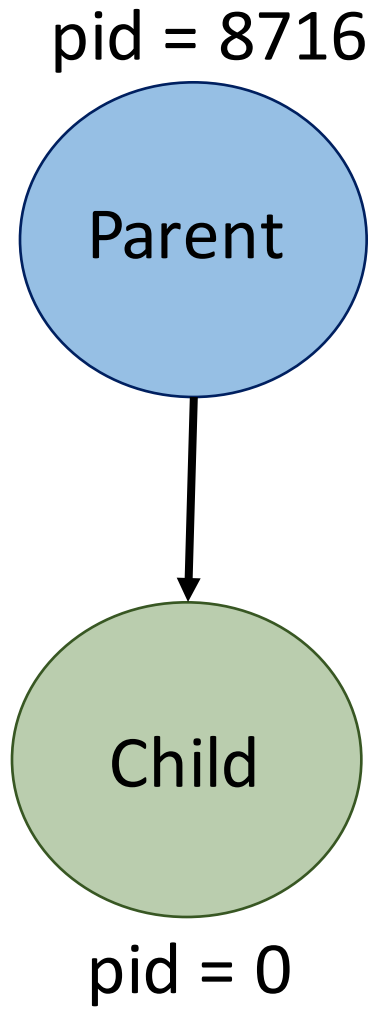
```
while (birthed++ < kids ) {
    fprintf(stderr, "%d: fork child\n", getpid());
    if ((pid = Fork()) == 0) {
        fprintf(stderr, "%d: I am the child\n", getpid());
        exit(0);
    }
    fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
}

while (died < kids) {
    pid = waitpid(-1, NULL, 0);
    fprintf(stderr,"%d: reap child %d\n", getpid(), pid);
    died++;
}
```



children-simple-wait.c (2)

Parent's process ID: 9812
Child's process ID: 8716

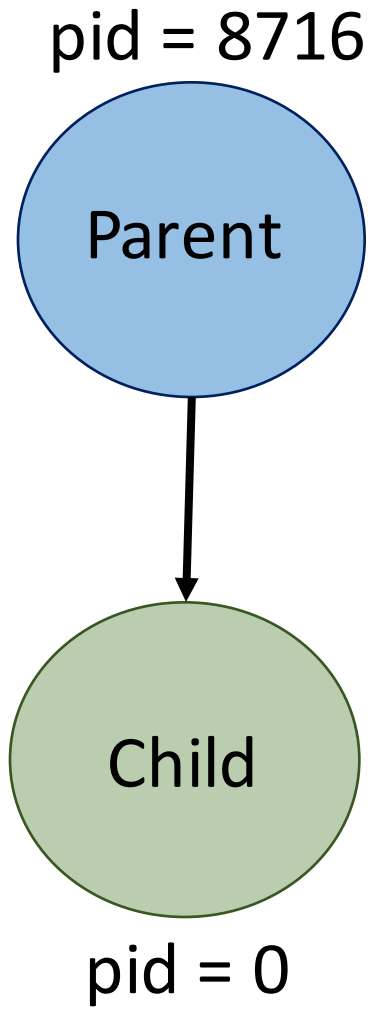
```
while (birthed++ < kids ) {
    fprintf(stderr, "%d: fork child\n", getpid());
    if ((pid = Fork()) == 0) {
        fprintf(stderr, "%d: I am the child\n", getpid());
        exit(0);
    }
    fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
}

while (died < kids) {
    pid = waitpid(-1, NULL, 0);
    fprintf(stderr,"%d: reap child %d\n", getpid(), pid);
    died++;
}
```



children-simple-wait.c (3)

```
while (birthed++ < kids ) {
    fprintf(stderr, "%d: fork child\n", getpid());
    if ((pid = Fork()) == 0) {
        fprintf(stderr, "%d: I am the child\n", getpid());
        exit(0);
    }
    fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
}

while (died < kids) {
    pid = waitpid(-1, NULL, 0);
    fprintf(stderr,"%d: reap child %d\n", getpid(), pid);
    died++;
}
```

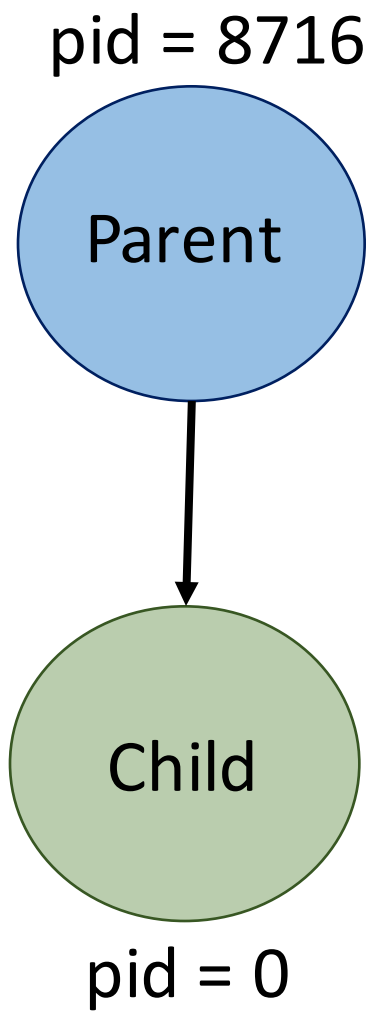
Parent is waiting for any of the children to finish (**hence the arg of -1 for process id**)



children-simple-wait.c (4)

Once child finishes……

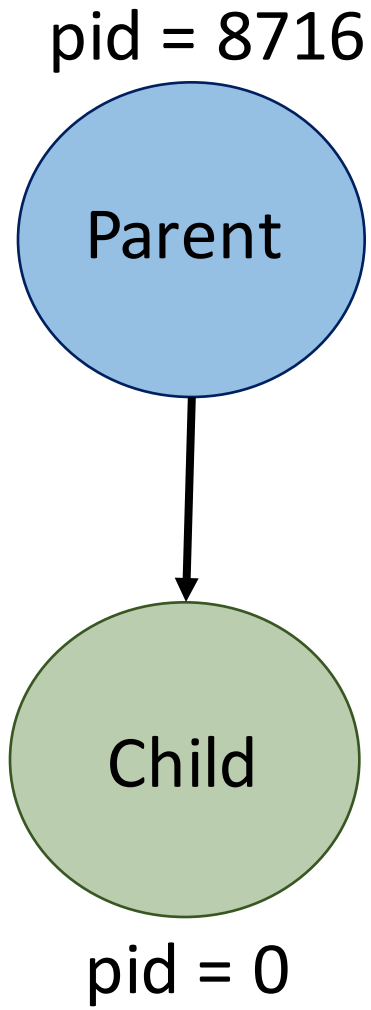
```
while (birthed++ < kids ) {
    fprintf(stderr, "%d: fork child\n", getpid());
    if ((pid = Fork()) == 0) {
        fprintf(stderr, "%d: I am the child\n", getpid());
        exit(0);
    }
    fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
}

while (died < kids) {
    pid = waitpid(-1, NULL, 0);
    fprintf(stderr,"%d: reap child %d\n", getpid(), pid);
    died++;
}
```



children-simple-wait.c (5)

```
while (birthed++ < kids ) {
    fprintf(stderr, "%d: fork child\n", getpid());
    if ((pid = Fork()) == 0) {
        fprintf(stderr, "%d: I am the child\n", getpid());
        exit(0);
    }
    fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
}

while (died < kids) {
    pid = waitpid(-1, NULL, 0);
    fprintf(stderr,"%d: reap child %d\n", getpid(), pid);
    died++;
}
```

Once child finishes……parent is unblocked and proceeds to **fprintf and died++**

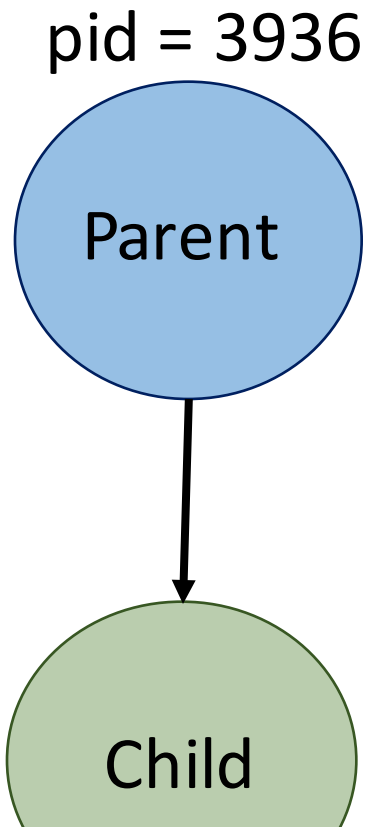


children-simple-wait.c (6)

```
while (birthed++ < kids ) {
    fprintf(stderr, "%d: fork child\n", getpid());
    if ((pid = Fork()) == 0) {
        fprintf(stderr, "%d: I am the child\n", getpid());
        exit(0);
    }
    fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
}

while (died < kids) {
    pid = waitpid(-1, NULL, 0);
    fprintf(stderr,"%d: reap child %d\n", getpid(), pid);
    died++;
}
```

SAMPLE OUTPUT:

```
3935: fork child
3935: Child is 3936
3936: I am the child
3935: reap child 3936
```

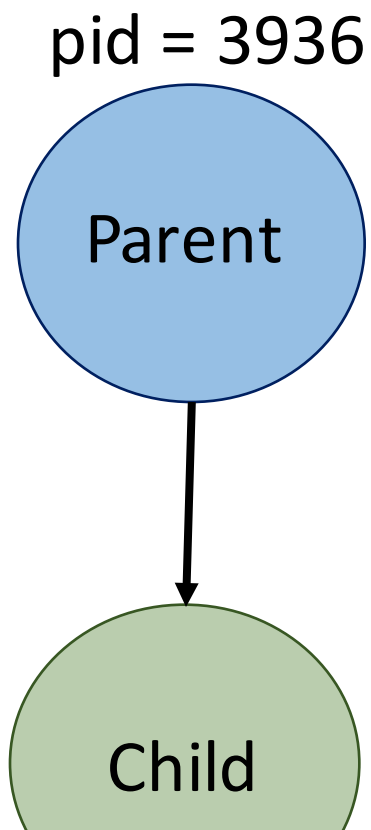


children-simple-wait.c (6)

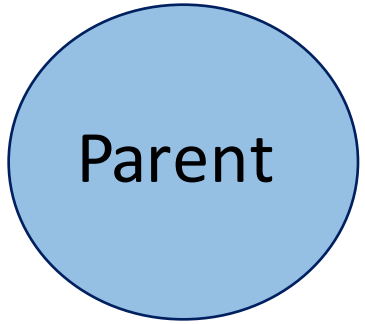
```
while (birthed++ < kids ) {
    fprintf(stderr, "%d: fork child\n", getpid());
    if ((pid = Fork()) == 0) {
        fprintf(stderr, "%d: I am the child\n", getpid());
        exit(0);
    }
    fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
}

while (died < kids) {
    pid = waitpid(-1, NULL, 0);
    fprintf(stderr,"%d: reap child %d\n", getpid(), pid);
    died++;
}
```

3935: fork child
3935: Child is 3936
3936: I am the child
3935: reap child 3936

Note:
"reap child…" must be printed after "I am the child".



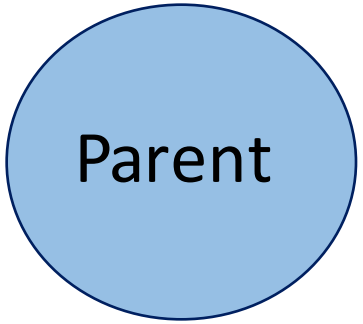
children-sigchild-no-wait.c




```
void handler(int sig)
 …….
while ((pid = waitpid(-1, NULL, 0)) > 0) { ……
}

void main () {
……..
    Signal(SIGCHLD, handler);
    while (birthed++ < kids ) {
      fprintf(stderr, "%d: fork child\n", getpid());
      if ((pid = Fork()) == 0) {
          fprintf(stderr, "%d: I am the child\n", getpid());
          exit(0);
      }
      fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
    }
```

children-sigchild-no-wait.c (2)

```
• void handler(int sig)
•  …….
• while ((pid = waitpid(-1, NULL, 0)) > 0) { ……
• }

• void main () {
…….
    Signal(SIGCHLD, handler);
•    while (birthed++ < kids ) {
•     fprintf(stderr, "%d: fork child\n", getpid());
•     if ((pid = Fork()) == 0) {
•        fprintf(stderr, "%d: I am the child\n", getpid());
•        exit(0);
•     }
•     fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
•    }
```



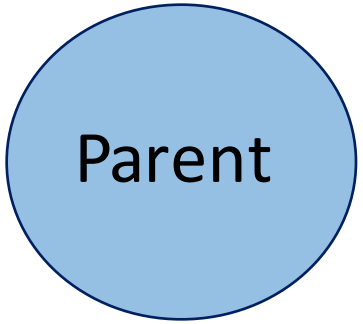
SIGCHLD signal:

A child process sends a SIGCHLD signal to the parent process when it finishes executing its code.

E.g. if a child executes exit(0) it will stop the program its running thus causing a SIGCHLD to be sent to its parent.

children-sigchild-no-wait.c (3)

```
void handler(int sig)
 .......
while ((pid = waitpid(-1, NULL, 0)) > 0) { ......
}

void main () {
.........
    Signal(SIGCHLD, handler);
    while (birthed++ < kids ) {
     fprintf(stderr, "%d: fork child\n", getpid());
     if ((pid = Fork()) == 0) {
        fprintf(stderr, "%d: I am the child\n", getpid());
        exit(0);
     }
     fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
    }
```

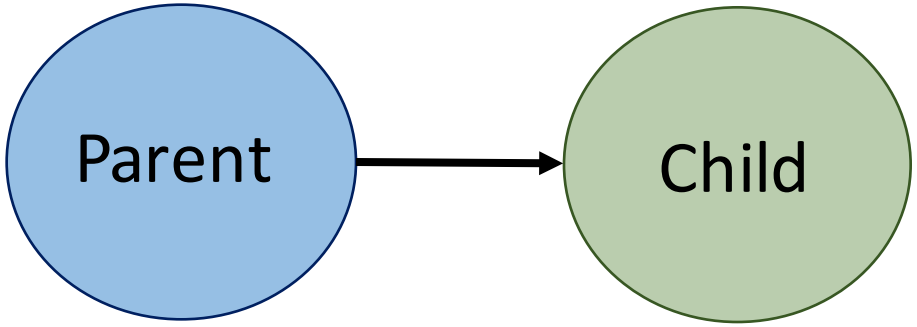


SIGCHLD signal:

1. Parent registers SIGCHLD signal to handler routine using Signal() function.

2. This means that when the parent receives a SIGCHLD signal from a child process it calls the handler() function where it reaps the child

children-sigchild-no-wait.c (4)



- void handler(int sig)
- …….
- while ((pid = waitpid(-1, NULL, 0)) > 0) { ……
- }

- void main () {

……..

```
    Signal(SIGCHLD, handler);
    while (birthed++ < kids ) {
      fprintf(stderr, "%d: fork child\n", getpid());
    if ((pid = Fork()) == 0) {
        fprintf(stderr, "%d: I am the child\n", getpid());
        exit(0);
      }
      fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
    }
```

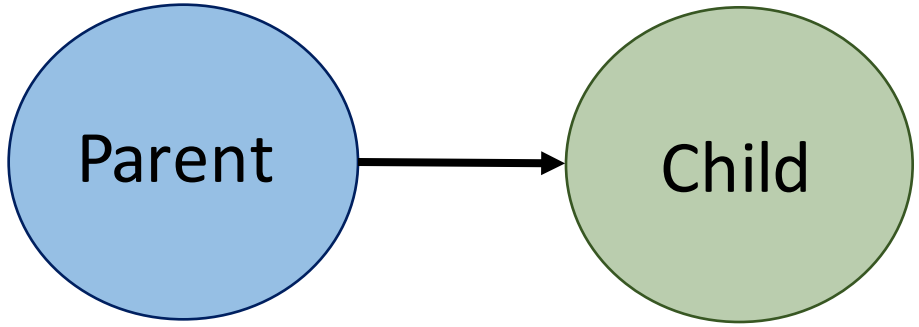
children-sigchild-no-wait.c (5)

- void handler(int sig)
- …….
- while ((pid = waitpid(-1, NULL, 0)) > 0) { ……
- }

- void main () {

……..

```
    Signal(SIGCHLD, handler);
    while (birthed++ < kids ) {
      fprintf(stderr, "%d: fork child\n", getpid());
    if ((pid = Fork()) == 0) {
        fprintf(stderr, "%d: I am the child\n", getpid());
        exit(0);
      }
      fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
    }
```



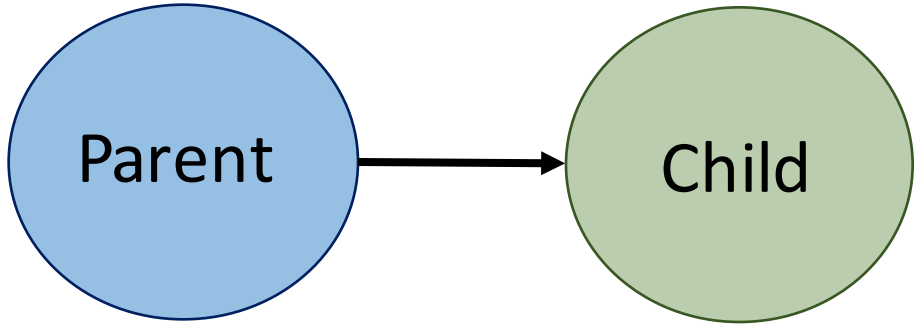
Parent needs to wait:

In this example, **the parent doesn't actually explicitly wait for the child**, and thus SIGCHLD handler may not be called.

If you **add the "pause"argument, the parent will pause for (any) signal to be delivered**. This happens to then let the children execute, which causeshandler to run.

children-sigchild-no-wait.c (6)

```
• void handler(int sig)
•  …….
• while ((pid = waitpid(-1, NULL, 0)) > 0) { ……
• }

• void main () {
…….
    Signal(SIGCHLD, handler);
    while (birthed++ < kids ) {
      fprintf(stderr, "%d: fork child\n", getpid());
    if ((pid = Fork()) == 0) {
•       fprintf(stderr, "%d: I am the child\n", getpid());
•       exit(0);
•     }
      fprintf(stderr, "%d: Child is %d\n", getpid(), pid);
    }
```



Parent needs to wait:

Thus we need a pause() or a wait() function that blocks the parent!

This way we can handle the SIGCHLD signal and reap the child.