

## Processes

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

1

## Processes

- What is a process?
- How do I make one? wait for one? kill one?
  - Birth
  - Life
  - Death

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

2

## Program or Process?

- Process
  - A process is the *context* (the information/data) maintained for an executing program
    - An executable instance of a program
  - A program can have many processes
  - Each process has a unique identifier
- Unix processes
  - Process #1 is known as the 'init' process (root of the process hierarchy)

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

3

## What makes up a Process?

- Program code
- Machine registers
- Global data
- Stack
- Open files
- An environment

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

4

## Process Context

- Process ID (`pid`)                    unique integer
- Parent process ID (`ppid`)            unique integer
- Current directory
- File descriptor table
- Environment                            `VAR=VALUE` pairs
- Pointer to program code
- Pointer to data                         Mem for global vars
- Pointer to stack                        Mem for local vars
- Pointer to heap                         Dynamically allocated memory
- Execution priority
- Signal information

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

5



## Unix Processes

- Address space
  - The address space is a section of memory that contains the code to execute as well as the process stack
- Set of data structures in the kernel to keep track of that process
  - Address space map
  - Current status of the process
  - Execution priority of the process
  - Resource usage of the process
  - Current signal mask
  - Owner of the process

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

6



## Process Lifetime

- Some processes run from system boot to shutdown
  - Servers & Daemons  
(e.g. Apache httpd server)
- Most processes come and go rapidly, as tasks start and complete
  - 'unit of work' on a modern computer
- A process can die a premature, even horrible death (say, due to a crash)

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

7



## Know your process

- Each process has a unique identifier

```
int myid = getpid();
```

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

8



## [ Know your process ]

- better...

```
pid_t myid = getpid()
```

- `pid_t: int` in linux,
- `pid_t: long` in other systems

- Know your parent

```
pid_t myparentid = getppid()
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

9



## [ Process Creation ]

- On creation, process needs resources
  - CPU, memory, files, I/O devices
- Get resources from the OS or from the parent process
  - Child process is restricted to a subset of parent resources
  - Prevents many processes from overloading system

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

10



## [ Process Creation ]

- Execution
  - Parent continues concurrently with child
  - Parent waits until child has terminated
- Address space
  - Child process is duplicate of parent process
  - Child process has a new program loaded into it

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

11



## [ Creating a Process – `fork()` ]

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

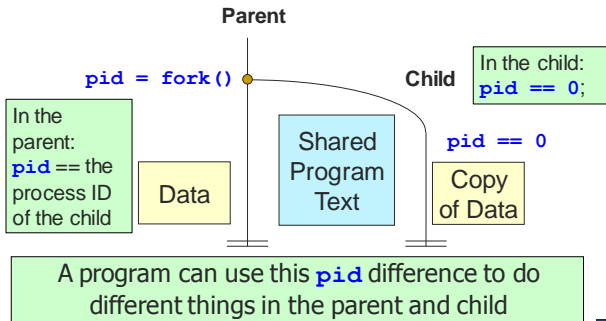
- Create a child process
  - The child is an (almost) exact copy of the parent
  - The new process and the old process both continue in parallel from the statement that follows the `fork()`
- Returns:
  - To child
    - 0 on success
  - To parent
    - process ID of the child process
    - -1 on error, sets `errno`

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

12



## [ Creating a Process – `fork()` ]



Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

13



## [ Creating a Process – `fork()` ]

- The child process is an exact copy of the parent process except
  - The child process has a unique process ID
  - The child process has a different parent process ID (i.e., the process ID of the calling process)
  - The child process has its own copy of the parent's file descriptors
- and some other stuff about memory and stuff that we'll learn later ...

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

14



## [ Example – `fork()` ]

```
int pid;
int status = 0;

if (pid = fork()) {
    /* parent */
    ...
    pid = wait(&status);
} else {
    /* child */
    ...
    exit(status);
}
```

`fork` returns twice:  
Parent: `pid ==` child process ID (`pid`)  
Child: `pid == 0`

Parent uses `wait` to sleep until the child exits.  
`wait` returns child `pid` and status.

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

15



## [ Example – `fork()` ]

Challenge:

write code so that child prints

```
'CHILD: my id is ___ and my parent id is ___'
```

and parent prints

```
'PARENT: my id is ___ and the child's id is ___'
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

16



## [ Example – fork () ]

```
childpid = fork()
```

What order will the output be printed in?

```
if (childpid == 0) {
    printf("CHILD: my id is %d and my parent id is %d.", getpid(), getppid());
    exit(0);
}
else {
    printf("PARENT: my id is %d and the child's id is %d.", childpid, getpid());
    exit(0);
}
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta



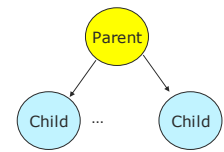
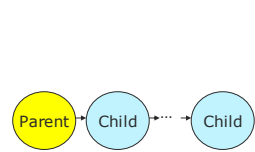
## [ Chain and Fan ]

### Chain

- Write code to make chain

### Fan

- Code to make N children of one parent process?



Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

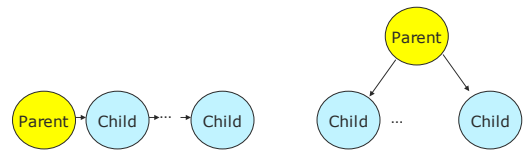


## [ Chain and Fan ]

### Chain

```
pid_t childpid = 0;
for (i=1;i<n;i++)
    if (childpid = fork())
        break;
```

### Fan



Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta



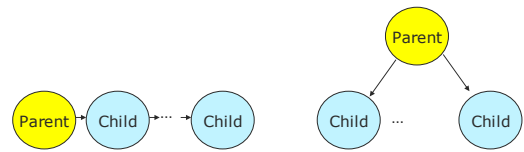
## [ Chain and Fan ]

### Chain

```
pid_t childpid = 0;
for (i=1;i<n;i++)
    if (childpid = fork())
        break;
```

### Fan

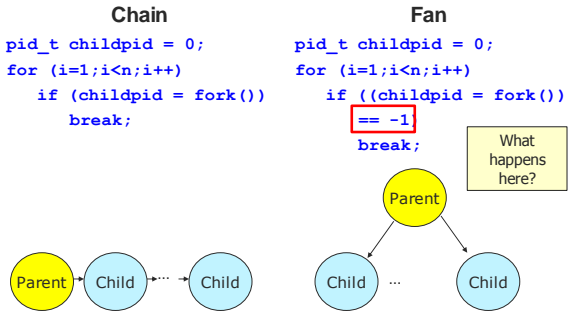
```
pid_t childpid = 0;
for (i=1;i<n;i++)
    if ((childpid = fork())
        <=0)
        break;
```



Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta



## Chain and Fan



Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

21



## Example – fork()

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

int main() {
  pid_t pid;          /* could be int */
  int i;
  pid = fork();
  if( pid > 0 ) { /* parent */
    for( i=0; i < 1000; i++ )
      printf("\t\t\tPARENT %d\n", i);
  }
}
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

22



## Example – fork()

```
else { /* child */
  for(i=0; i < 1000; i++)
    printf( "CHILD %d\n", i );
}
return 0;
}
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

23



## Example – fork() Possible Output

```
CHILD 0
CHILD 1
CHILD 2

PARENT 0
PARENT 1
PARENT 2
PARENT 3

CHILD 3
CHILD 4

PARENT 4

:
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

24



## Example – `fork()` Notes

- `i` is copied between parent and child
- Switching between parent and child depends on many factors
  - Machine load, system process scheduling
- I/O buffering effects amount of output shown
- Output interleaving is nondeterministic
  - Cannot determine output by looking at code

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

25



## Waiting for a child to finish – `wait()`

```
#include <sys/types.h>
#include <wait.h>
pid_t wait(int *status);
```

- Suspend calling process until child has finished
- Returns:
  - Process ID of terminated child on success
  - -1 on error, sets `errno`
- Parameters:
  - `status`: status information set by `wait` and evaluated using specific macros defined for `wait`.

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

26



## Waiting for any child to finish

```
#include <errno.h>
#include <sys/wait.h>

pid_t childpid;

childpid = wait(NULL);
if (childpid != -1)
    printf("waited for child with pid %ld\n",
           childpid);
```

(see "man 2 wait")

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

27



## `wait()` Function

- allows parent process to wait (block) until child finishes
- causes the caller to suspend execution until child's status is available

<code>errno</code>	cause
<code>ECHILD</code>	Caller has no unwaited-for children
<code>EINTR</code>	Function was interrupted by signal
<code>EINVAL</code>	Options parameter of <code>waitpid</code> was invalid

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

28



## Waiting for a child to finish – `waitpid()`

```
#include <sys/types.h>
#include <wait.h>
pid_t waitpid(pid_t pid, int *status, int
options);
```

- Suspend calling process until child specified by `pid` has finished
- Returns:
  - Process ID of terminated child on success
  - 0 if `WNOHANG` and no child available, sets `errno`
  - -1 on error, sets `errno`
- Parameters:
  - `status`: status information set by `wait` and evaluated using specific macros defined for `wait`.

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

29



## Waiting for a child to finish – `waitpid()`

```
#include <sys/types.h>
#include <wait.h>
pid_t waitpid(pid_t pid, int *status, int
options);
```

- Suspend calling process until child specified by `pid` has finished
- Parameters:
  - `pid`:
    - <-1: wait for any child process whose process group ID is equal to the absolute value of `pid`.
    - -1 wait for any child process (same as `wait`)
    - 0 wait for any child process whose process group ID is equal to that of the calling process.
    - > 0 wait for the child whose process ID is equal to the value of `pid`.

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

30



## Waiting for a child to finish – `waitpid()`

```
#include <sys/types.h>
#include <wait.h>
pid_t waitpid(pid_t pid, int *status, int
options);
```

- Suspend calling process until child specified by `pid` has finished
- Parameters:
  - `options`:
    - `WNOHANG`: return immediately if no child has exited.
    - `WUNTRACED`: return for children which are stopped, and whose status has not been reported.

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

31



## When good processes die

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

32



## Process Termination

- Upon completion of last statement
  - A process automatically asks the OS to delete it
  - All of the child's resources are de-allocated
  - Child process may return output to parent process
- Other termination possibilities: Aborted by parent process
  - Child has exceeded its usage of some resources
  - Task assigned to child is no longer required
  - Parent is exiting and OS does not allow child to continue without parent

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

33



## Process Termination

- Voluntary termination
  - Normal exit
    - End of `main()`
  - Error exit
    - `exit(2)`
- Involuntary termination
  - Fatal error
    - Divide by 0, core dump / seg fault
  - Killed by another process
    - `kill` proclD, end task

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

34



## How to List all Processes?

- On Windows: run Windows task manager
  - Hit Control+ALT+delete
  - Click on the "processes" tab
- On UNIX
  - `> ps -e` also, `pstree`
  - Try "`man ps`"

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

35



## Example – `fork()`

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

int main() {
    pid_t pid;          /* could be int */
    int i;
    pid = fork();
    if( pid > 0 ) {     /* parent */
        for( i=0; i < 1000; i++ )
            printf("\t\t\tPARENT %d\n", i);
    }
    else { /* child */
        for(i=0; i < 1000; i++)
            printf( "CHILD %d\n", i );
    }
    return 0;
}
```

How can you use `ps` to see the processes that are created?

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

36



## Example – `fork()`

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

int main() {
    pid_t pid;          /* could be int */
    int i;
    pid = fork();
    if (pid > 0) {      /* parent */
        for(i=0; i < 1000; i++)
            printf("\t\t\tPARENT %d\n", i);
    }
    else { /* child */
        for(i=0; i < 1000; i++)
            printf("CHILD %d\n", i);
    }
    return 0;
}
```

How can you use `ps` to see the processes that are created?

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

37



## System view of processes (Next)

- 5 state Process Model
- Process Control Block
- Context Switch

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

38

