# Processes appendix

# 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)




# 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

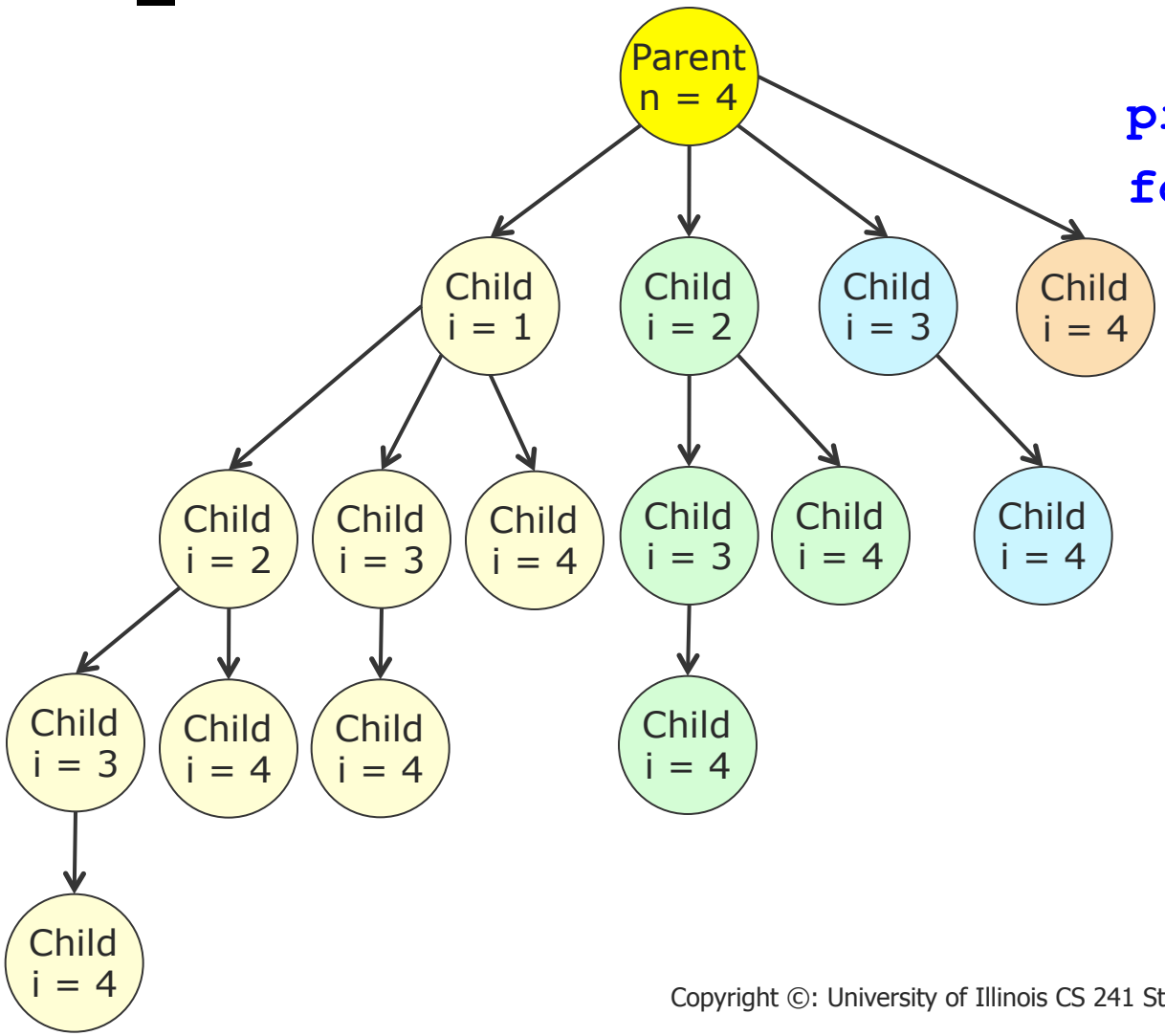



# Process Creation

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




# Chain and Fan Example (n=4)




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





# Orphan Example

```
void fork8() {
    if (fork() == 0) {
        /* Child */
        printf("Running Child, PID = %d\n",
               getpid());
        while (1); /* Infinite loop */
    } else {
        printf("Terminating Parent, PID = %d\n",
               getpid());
        exit(0);
    }
}
```




# Orphan Example

```
void fork8() {
    if (fork() == 0) {
        /* Child */
        printf("Running Child
               getpid());
        while (1); /* Infinit
    } else {
        printf("Terminating P
               getpid());
        exit(0);
    }
}
```

```
Linux> ./forktest 8
Running Child, PID = 9413
Terminating Parent, PID = 9412
Linux> ps
  PID TTY          TIME CMD
 9413 pts/1    00:00:07 forktest
 9416 pts/1    00:00:00 ps
29160 pts/1    00:00:00 bash
Linux> kill 9413
Linux> ps
  PID TTY          TIME CMD
 9422 pts/1    00:00:00 ps
29160 pts/1    00:00:00 bash
```

- Child process still active even though parent has terminated
- Must kill explicitly, or else will keep running indefinitely




# wait(), waitpid() System Calls

- If status is not NULL, wait() stores status information in the int to which it points. This integer can be inspected with specific macros (see man pages):

- WIFEXITED(status)
  - returns true if the child terminated normally, that is, by calling exit, or by returning from main().

- WEXITSTATUS(status)
  - returns the exit status of the child. This consists of the least significant 8 bits of the status argument that the child specified in a call to exit or as the argument for a return statement in main(). This macro should only be employed if WIFEXITED returned true.




# 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
  - -1 on error, sets **`errno`**
- Parameters:
  - **`status`**: status information set by **`wait`** and evaluated using specific macros defined for **`wait`**.




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

```
void fork11() {
    pid_t pid[N];
    int i;
    int child_status;

    for (i = 0; i < N; i++)
      if ((pid[i] = fork()) == 0)
        exit(100+i); /* Child */
    for (i = N-1; i >= 0; i--) {
        pid_t wpid = waitpid(pid[i], &child_status, 0);
        if (WIFEXITED(child_status))
          printf("Child %d terminated with exit status %d\n",
            wpid, WEXITSTATUS(child_status));
        else
          printf("Child %d terminated abnormally\n", wpid);
    }
}
```





# 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`”