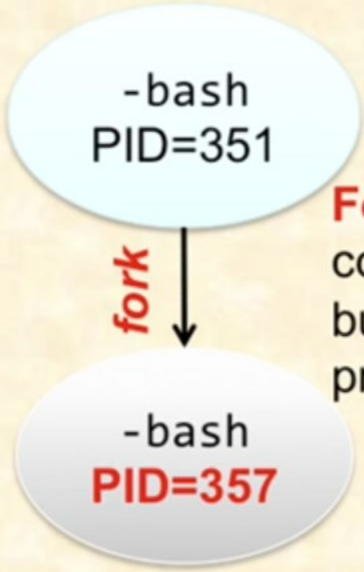
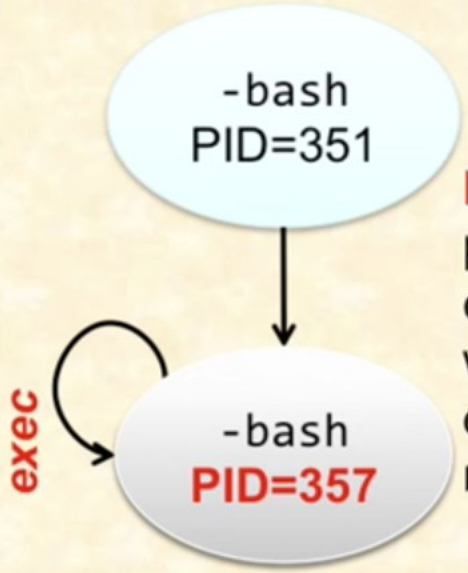
Fork & Exec in Linux

- Running a new program on Linux involves 2 steps
  1. **Fork**: Clone current process
  2. **Exec**: Replace current program with a different one



Fork: Identical copy of parent but different process



Exec: Replace program of current process with a different one. **PID does not change!**

Fork

- Fork is a system call used to clone a running process
  - The cloned process is a child of the parent

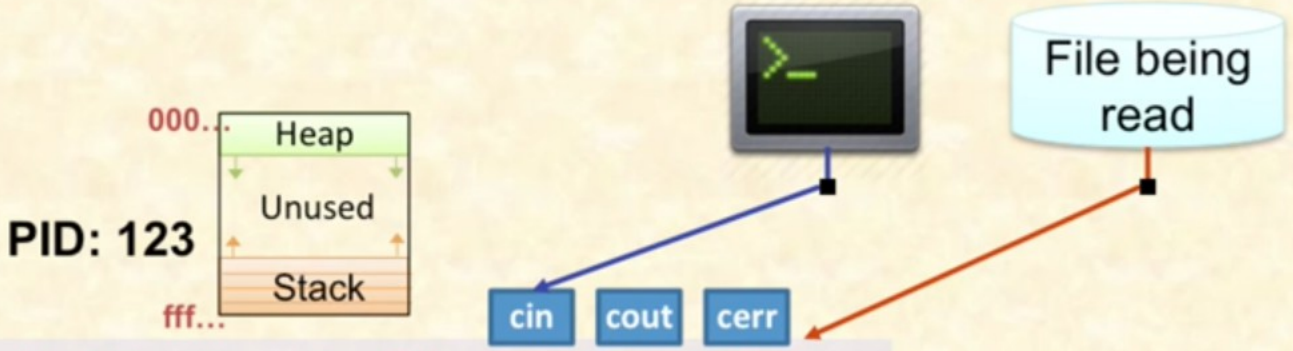
Fork

- Fork is a system call used to clone a running process
  - The cloned process is a child of the parent
- The parent and child are identical processes
  - Have exactly the same stack (sequence of function calls)
  - Have same virtual memory (so pointers and other data is preserved)
  - Have same set of files and consequently have
    - **Same set of open files, devices, sockets, etc.**

Fork

- Fork is a system call used to clone a running process
  - The cloned process is a child of the parent
- The parent and child are identical processes
  - Have exactly the same stack (sequence of function calls)
  - Have same virtual memory (so pointers and other data is preserved)
  - Have same set of files and consequently have
    - **Same set of open files, devices, sockets, etc.**
- The cloned child differs from the parent in:
  - The process ID (`pid`)
  - The return value from `fork` in the parent and child a slightly different
    - This is used to tell the difference between parent and child process

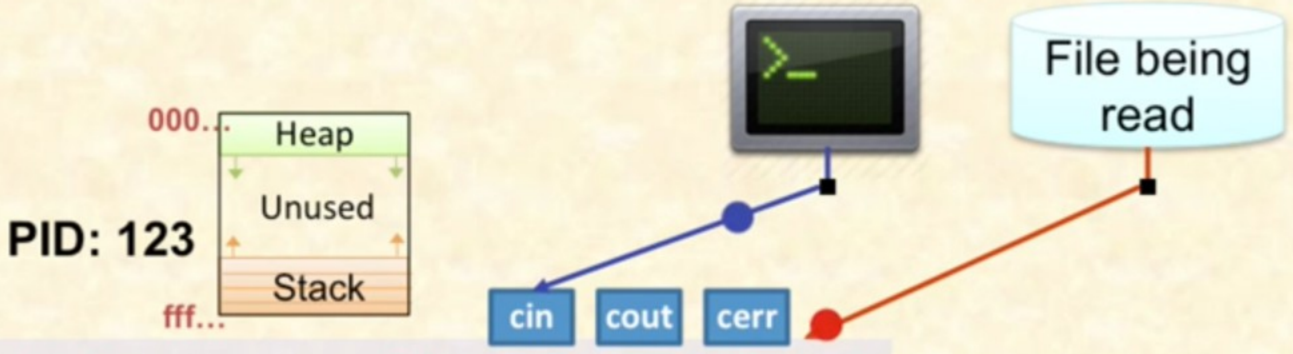
Fork in action



```
// Assume more lines before this
cout << "Forking...\n";
int pid = fork();
cout << "Forked!\n";
cout << "pid = " << pid << ".\n";
// Assume more lines after this
```

Output:

Fork in action

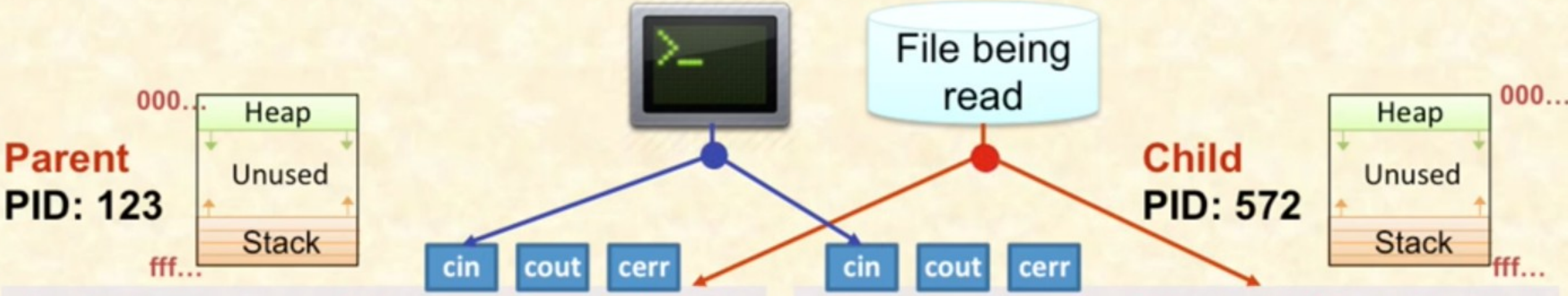


```
// Assume more lines before this
cout << "Forking...\n";
int pid = fork();
cout << "Forked!\n";
cout << "pid = " << pid << ".\n";
// Assume more lines after this
```

Output:

```
Forking...
```

Fork in action



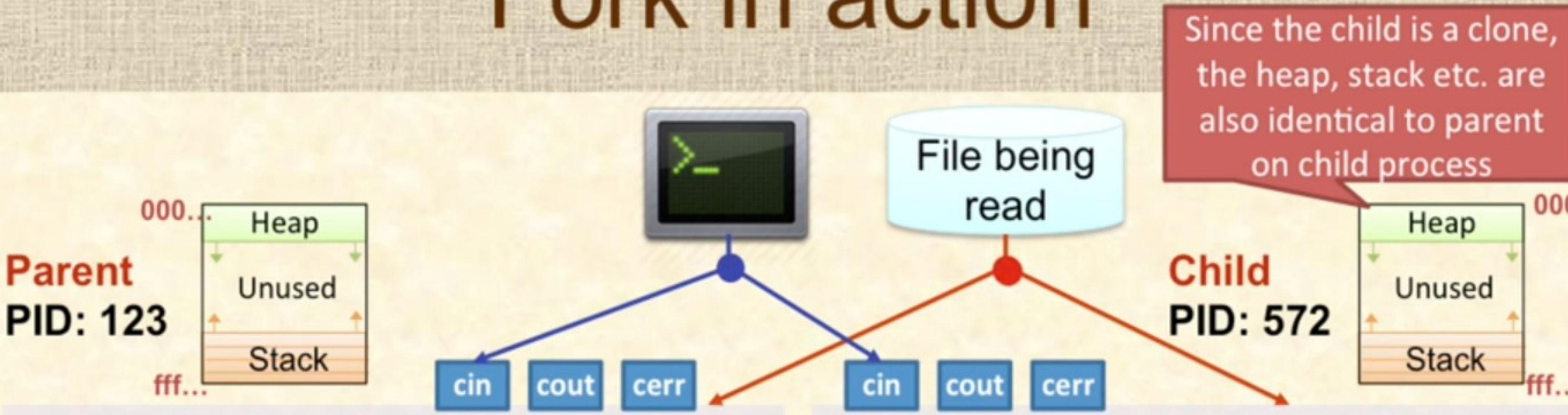
```
// Assume more lines before this
cout << "Forking...\n";
int pid = fork();
cout << "Forked!\n";
cout << "pid = " << pid << ".\n";
// Assume more lines after this
```

```
// Assume more lines before this
cout << "Forking...\n";
int pid = fork();
cout << "Forked!\n";
cout << "pid = " << pid << ".\n";
// Assume more lines after this
```

Output:

```
Forking...
```

Fork in action



```
// Assume more lines before this
cout << "Forking...\n";
int pid = fork();
cout << "Forked!\n";
cout << "pid = " << pid << ".\n";
// Assume more lines after this
```

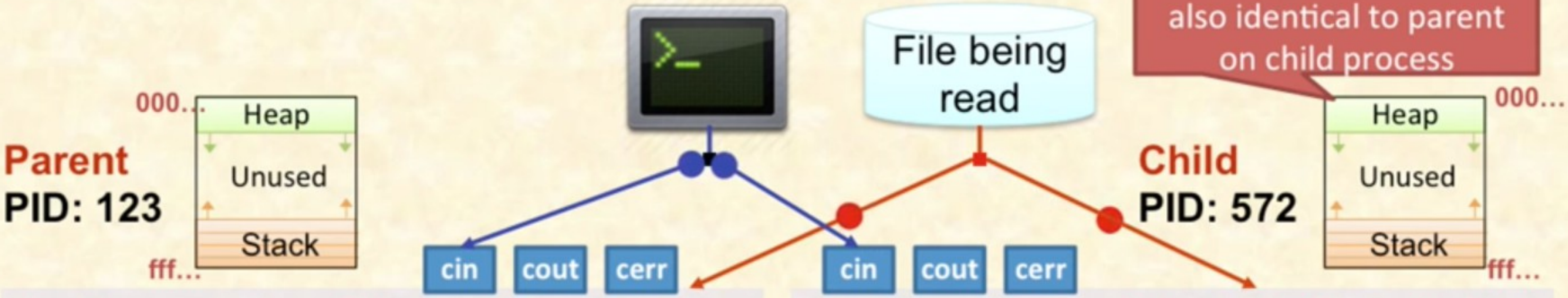
```
// Assume more lines before this
cout << "Forking...\n";
int pid = fork();
cout << "Forked!\n";
cout << "pid = " << pid << ".\n";
// Assume more lines after this
```

Output:

```
Forking...
```

Fork in action

Since the child is a clone, the heap, stack etc. are also identical to parent on child process



```
// Assume more lines before this
cout << "Forking...\n";
int pid = fork();
cout << "Forked!\n";
cout << "pid = " << pid << ".\n";
// Assume more lines after this
```

```
// Assume more lines before this
cout << "Forking...\n";
int pid = fork();
cout << "Forked!\n";
cout << "pid = " << pid << ".\n";
// Assume more lines after this
```

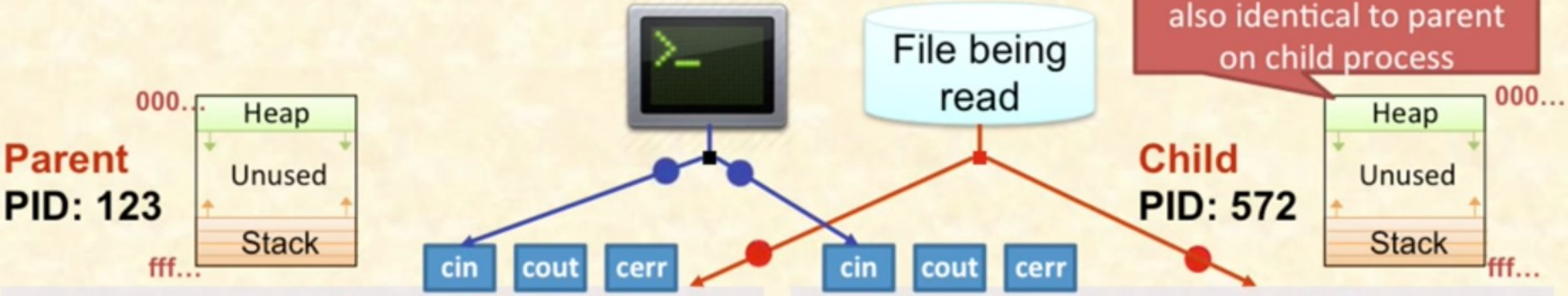
Output:

```
Forking...
Forked!
Forked!
```

Note that these outputs can appear in any order or could even be intermixed

Fork in action

Since the child is a clone, the heap, stack etc. are also identical to parent on child process



```
// Assume more lines before this
cout << "Forking...\n";
int pid = fork();
cout << "Forked!\n";
cout << "pid = " << pid << ".\n";
// Assume more lines after this
```

```
// Assume more lines before this
cout << "Forking...\n";
int pid = fork();
cout << "Forked!\n";
cout << "pid = " << pid << ".\n";
// Assume more lines after this
```

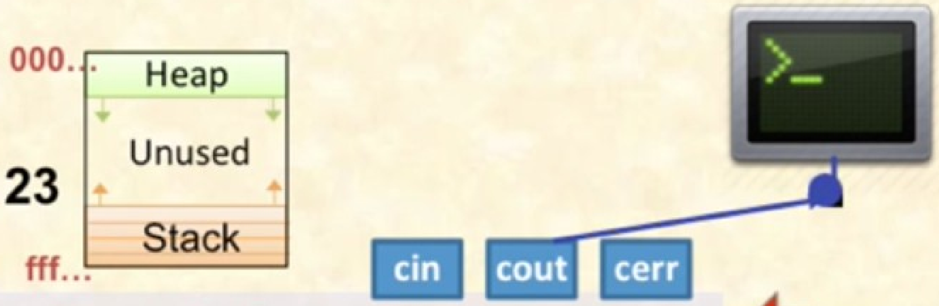
Output:

```
Forking…
Forked!
Forked!
pid = 572
pid = 0
```

Note that these outputs can appear in any order or could even be intermixed

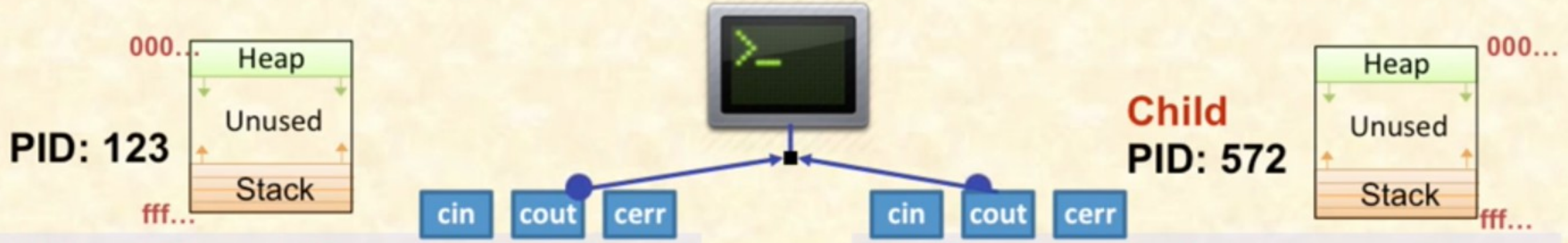
Return value of fork is:
- 0 on child
- Actual pid of child in parent

Fork + Exec in action



```
// More lines before…
int pid = fork();
if (pid == 0) {
  // Child runs a program
  execlp("ps", "ps", nullptr);
} else {
  // Parent waits for
  // child to finish
  waitpid(pid, nullptr, 0);
}
// More lines after…
```

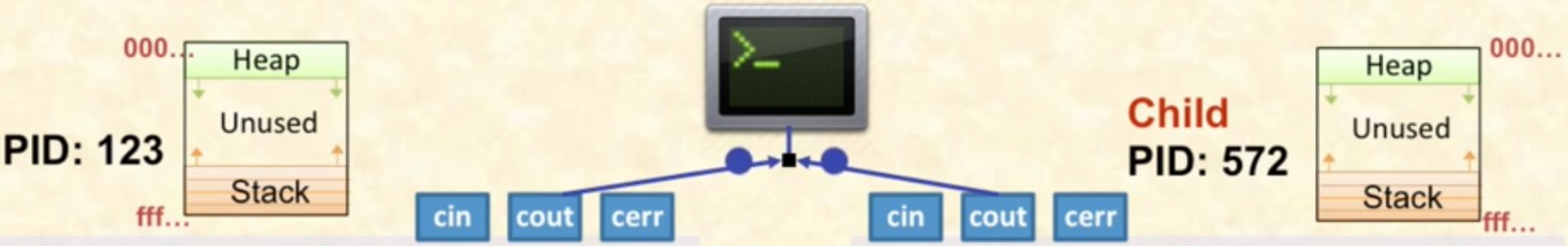
Fork + Exec in action



```
// More lines before…
int pid = fork();
if (pid == 0) {
  // Child runs a program
  execlp("ps", "ps", nullptr);
} else {
  // Parent waits for
  // child to finish
  waitpid(pid, nullptr, 0);
}
// More lines after…
```

```
// More lines before…
int pid = fork();
if (pid == 0) {
  // Child runs a program
  execlp("ps", "ps", nullptr);
} else {
  // Parent waits for
  // child to finish
  waitpid(pid, nullptr, 0);
}
// More lines after…
```

Fork + Exec in action



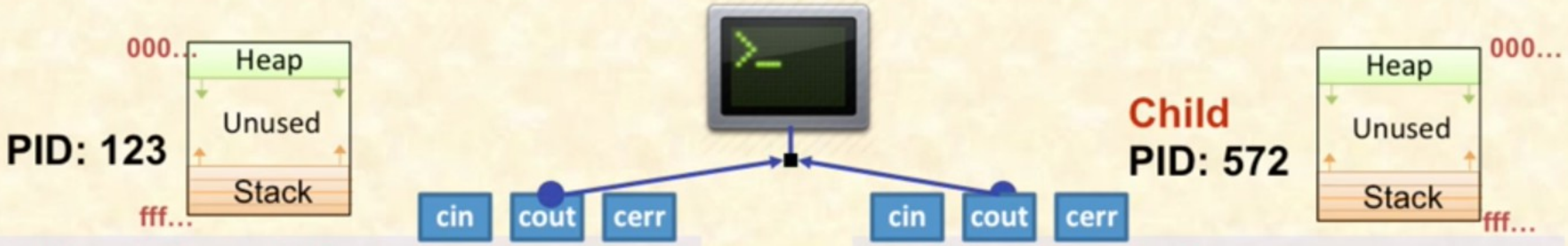
```
// More lines before…
int pid = fork();
if (pid == 0) {
  // Child runs a program
  execlp("ps", "
} else {
  // Parent waits for
  // child to finish
  waitpid(pid, nullptr, 0);
}
// More lines after…
```

fork returns:
- Actual pid of child here in parent

```
// More lines before…
int pid = fork();
if (pid == 0) {
  // Child runs a program
  execlp("ps", "
} else {
  // Parent waits for
  // child to finish
  waitpid(pid, nullptr, 0);
}
// More lines after…
```

fork returns:
- 0 in child process

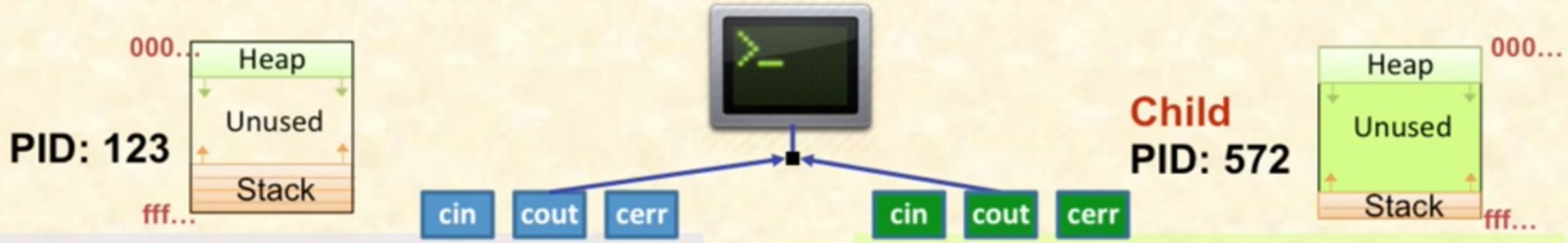
Fork + Exec in action



```
// More lines before…
int pid = fork();
if (pid == 0) {
  // Child runs a program
  execlp("ps", "ps", nullptr);
} else {
  // Parent waits for
  // child to finish
  waitpid(pid, nullptr, 0);
}
// More lines after…
```

```
// More lines before…
int pid = fork();
if (pid == 0) {
  // Child runs a program
  execlp("ps", "ps", nullptr);
} else {
  // Parent waits for
  // child to finish
  waitpid(pid, nullptr, 0);
}
// More lines after…
```

Fork + Exec in action



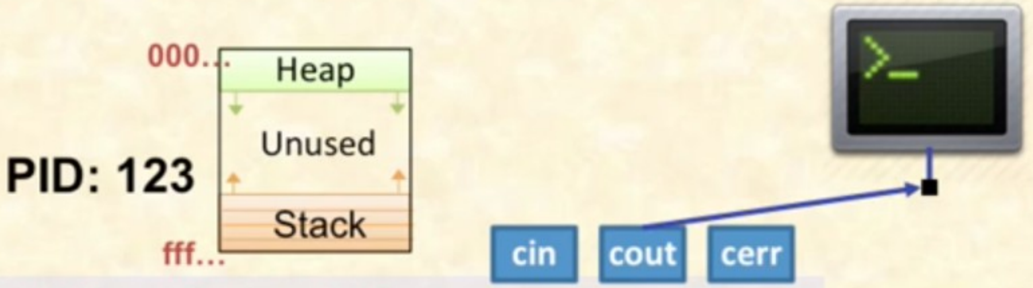
```
// More lines before…
int pid = fork();
if (pid == 0) {
  // Child runs a program
  execlp("ps", "ps", nullptr);
} else {
  // Parent waits for
  // child to finish
  waitpid(pid, nullptr, 0);
}
// More lines after…
```

```
// Program for ps command
int main() {
  // Program for ps command is
  // Loaded from disk to replace
  // the currently running program



}
```

Fork + Exec in action



```
// More lines before…
int pid = fork();
if (pid == 0) {
  // Child runs a program
  execlp("ps", "ps", nullptr);
} else {
  // Parent waits for
  // child to finish
  waitpid(pid, nullptr, 0);
}
// More lines after…
```

Recap of Fork + Exec

Fork + Exec system calls are used to run new processes
- **Fork**: Clones the current process to create a new process
- **Exec**: Replaces current program with a another one

Forking processes has advantages
- Threads do not enjoy many of these advantages

1. Fault tolerance: If child process crashes parent process is unaffected Heavily used in web-servers & databases to improve resilience
2. Security: Parent and child can have different security settings (or even run as different users) for better auditing
3. Low latency: Since memory and I/O streams are identical start-up overheads are minimized.
4. Concurrency & Performance: Parent and child can collaborate to perform different operations to improve performance.
5. Resource utilization: Processes get full set of resources enabling effective utilization of computational platforms
   - Easier to migrate processes across cloud infrastructures
6. Monitoring & debugging: Independent processes are easier to monitor, debug, and malicious processes can be killed.