

osj

April 9, 2025

# Contents

0.1	Introduction . . . . .	2
-----	------------------------	---

## 0.1 Introduction

Process creation & termination • One process (the “parent”) can create another (the “child”) – A new PCB is allocated and initialized – Homework: run ‘ps auxwww’ in the shell; PPID is the parent’s PID • In POSIX, child process inherits most of parent’s attributes – UID, open files (should be closed if unneeded; why?), cwd, etc. • While executing, PCB moves between different queues – According to state change graph – Queues: runnable, sleep/wait for event i (i=1,2,3...) • After a process dies (exit()s / interrupted), it becomes a zombie – Parent uses wait\* syscall to clear zombie from the system (why?) – Wait syscall family: wait, waitpid, waitid, wait3, wait4; example: – pid\_t wait4(pid\_t, int \*wstatus, int options, struct rusage \*rusage); • Parent can sleep/wait for its child to finish or run in parallel – wait\*() will block unless WNOHANG given in ‘options’ – Homework: read ‘man 2 wait’ OS (234123) - processes & signals 7

```
int main(int argc, char *argv[]) int pid = fork(); if( pid==0 ) // // child // printf("parent=getppid(),
getpid()); else if( pid > 0 ) // // parent // printf("parent=getpid(), pid); else // print
string associated // with errno perror("fork() failed"); return 0; • fork() initializes a new
PCB – Based on parent’s value – PCB added to runnable queue • Now there are 2 processes
– At same execution point • Child’s new address space – Complete copy of parent’s space,
with one difference... • fork() returns twice – At the parent, with pid>0 – At the child, with
pid=0 • What’s the printing order? • ‘errno’ – a global variable – Holds error num of last
syscall OS (234123) - processes & signals 8 fork() – spawn a child process
```