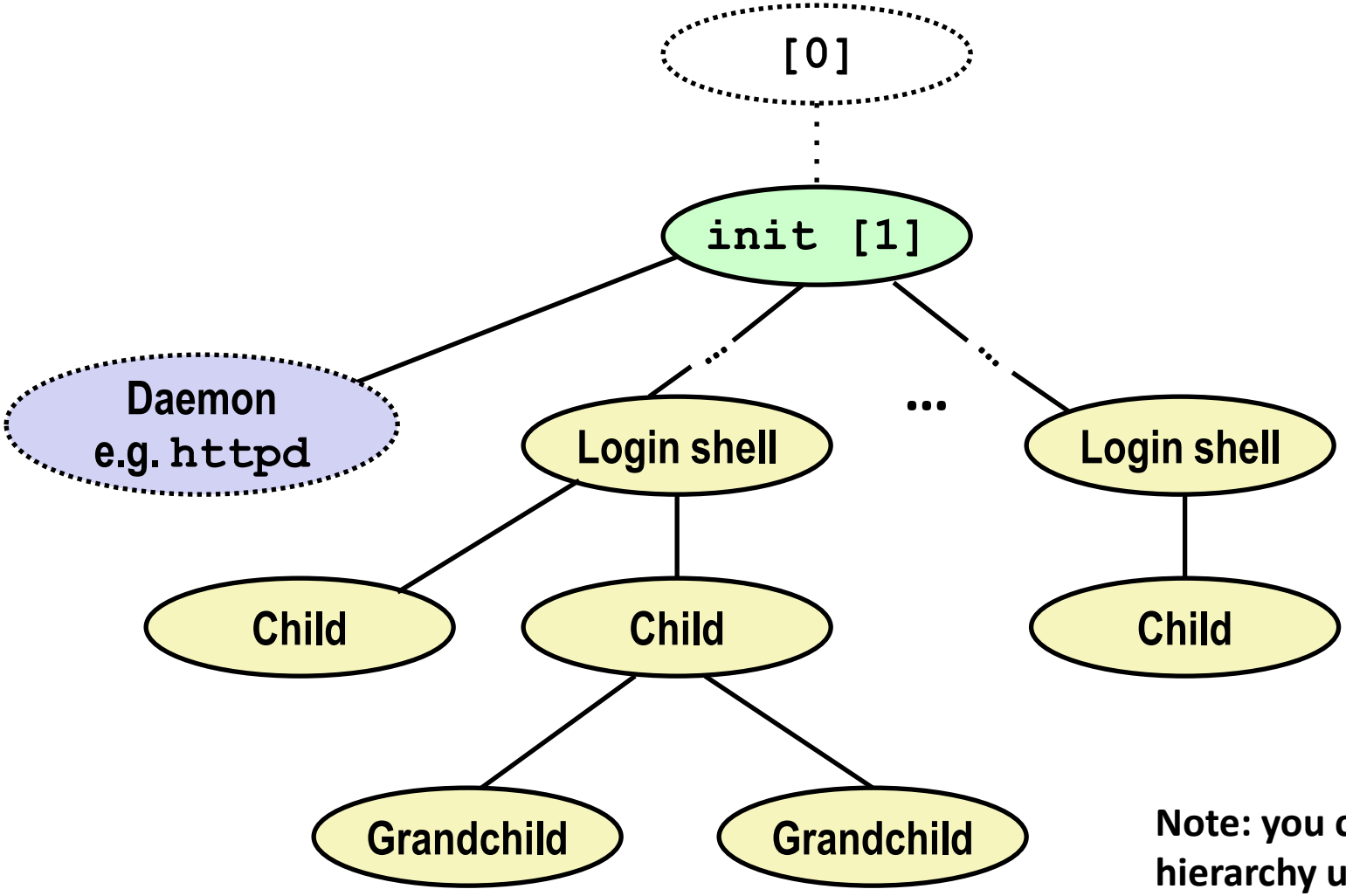
Exceptional Control Flow:
Shells & Forks

Linux Process Hierarchy



Note: you can view the hierarchy using the Linux `pstree` command

Shell Programs

- A *shell* is an application program that runs programs on behalf of the user.
  - `sh` Original Unix shell (Stephen Bourne, AT&T Bell Labs, 1977)
  - `csh/tcsh` BSD Unix C shell
  - `bash` "Bourne-Again" Shell (default Linux shell)

```
int main()
{
    char cmdline[MAXLINE]; /* command line */

    while (1) {
        /* read */
        printf("> ");
        Fgets(cmdline, MAXLINE, stdin);
        if (feof(stdin))
            exit(0);

        /* evaluate */
        eval(cmdline);
    }
}
```

shellex.c

Execution is a sequence of read/evaluate steps

Simple Shell `eval` Function

```c
void eval(char *cmdline)
{
    char *argv[MAXARGS]; /* Argument list execve() */
    char buf[MAXLINE];   /* Holds modified command line */
    int bg;              /* Should the job run in bg or fg? */
    pid_t pid;           /* Process id */

    strcpy(buf, cmdline);
    bg = parseline(buf, argv);
    if (argv[0] == NULL)
        return;   /* Ignore empty lines */

    if (!builtin_command(argv)) {
        if ((pid = Fork()) == 0) {   /* Child runs user job */
            if (execve(argv[0], argv, environ) < 0) {
                printf("%s: Command not found.\n", argv[0]);
                exit(0);
            }
        }

        /* Parent waits for foreground job to terminate */
        if (!bg) {
            int status;
            if (waitpid(pid, &status, 0) < 0) /* Reaping */
                unix_error("waitfg: waitpid error");
        }
        else
            printf("%d %s", pid, cmdline);
    }
    return;
}
```

shellex.c

Problem with Simple Shell Example

- **Our example shell correctly waits for and reaps foreground jobs**

- **But what about background jobs?**
  - Will become zombies when they terminate
  - Will never be reaped because shell (typically) will not terminate
  - Will create a memory leak that could run the kernel out of memory

ECF to the Rescue!

- **Solution: Exceptional control flow**
  - The kernel will interrupt regular processing to alert us when a background process completes
  - In Unix, the alert mechanism is called a ***signal***