Lecture 33 (4-17-26)

Processes / Signals (System Calls)

Review on Processes (important for final)

System calls and their actions

- `fork()` → allocates a new process
- `exec()` → load new program code
  - `exec()` can fail if :
    - the program doesn't exist
    - the program is not executable
- `exit()` → quit and set exit status
- `wait()` → retrieve exit status and deallocate
- Why is it that the parent waits and the child execs, and not the other way around?
- Whats a tool to see the kernel state:
  - `ps`

Term and description (need to be able to identify these in code)

- **orphan** → Child process when parents dies (terminates) before it waits
- **zombie** → Dead child process (terminated) that hasn't been waited on yet (parent can be alive or dead)
- **fork bomb** → excessive/uncontrolled forking
- you can have **orphan-zombies**:
  - thank about when the parent dies, and then later the child dies. In this case the child has died and hasn't been waited on yet.
- remember if every process does the exact same thing, that is DATA PARALLELISM

Signals

- you can ignore children instead of waiting for them
  - `signal(SIGCHLD, SIG_IGN)`
  - this automatically sends them to adoption

- You can also handle the `SIGCHLD` signal manually
- ```
  voud sigchld_handler(int signum){
    int status;
    pid_t pid = wait(&status);

    printf("[Parent POV] Child pid: %d, Child status: %d\n", pid, status);
  }


  int main(){

    signal(SIGCHLD, sigchld_handler)
    //event based programming
    //if this event happens, stop what I'm doing, run this code:
  sigchld_handler, and then go back to where I was
  }
  ```

- remember if a child changes a variable it does not impact the parent (recall `n_children` example from class)
  - so this is a case where you may want to use `global` variables

---

`sigalarm.c`

```
/* sigalrm.c: An example of using alarm */

#include <errno.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <unistd.h>

bool Wakeup = false;
```
```

```c
void sigalrm_handler(int signum) {
    Wakeup = true;
}

int main(int argc, char *argv[]) {
    int timeout = atoi(argv[1]);
    int seconds = 1;

    // Register alarm
    struct sigaction action = {.sa_handler = sigalrm_handler};
    sigaction(SIGALRM, &action, NULL);

    // Setup alarm
    alarm(timeout);

    // Print out message while waiting for alarm to trigger
    while (!Wakeup) {
         printf("\r%d. Waiting...", seconds++);
         fflush(stdout);
         sleep(1);
    }

    // Print out alarm
    printf("\r!!! WAKE UP !!!\n");

    return EXIT_SUCCESS;
}

/* vim: set sts=4 sw=4 ts=8 expandtab ft=c: */
```