15-213 Gremlin

Intro to Race Conditions

&

The Shell Lab

What is the Gremlin?

- A Fall ’06 Exam 2 Problem
- Tests race conditions
- Tests understanding of basic unix system calls.
- Tests understanding of process groups receiving signals.
- Turned out to be less straight-foreword than intended.

Intro Unix System Calls

- System Calls
  - pid_t fork();
    - Creates child process.
    - Parent returns pid_t of child. Child returns 0.
    - Non-positive on error.
  - void kill(pid_t pid, int sig);
    - Sends sig to pid if pid > 0.
    - Otherwise sends to every process in process group.

Intro System Calls (Cont)

- sighandler_t signal(int signum, sighandler_t handler);
  - typedef void (*sighandler_t)(int);
  - Registers function handler to be called when process receives signum signal.
- pid_t setpgid(pid_t pid, pid_t pgid);
  - Set group id of pid to pgid.
  - If pid==pgid==0 then put the current process in a new group where its gid = its pid.
- void exit(int status)
  - Exit from program with exit status as status.

What does this program output?

```
int val = 3;

void Exit(int val)
{
  printf("%d", val);
  exit(0);
}

void usr1_handler(int sig)
{
  Exit(val);
}
```

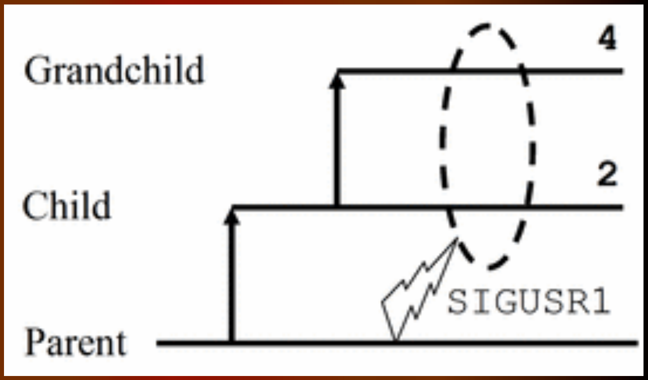
```
int main()
{
  int pid;
  signal(SIGUSR1, usr1_handler);

  if ((pid = fork()) == 0)
  {
    setpgid(0, 0);
    if (fork())
      Exit(val + 1);
    else
      Exit(val - 1);
  }

  kill(-pid, SIGUSR1);
}
```

How many outputs did you find?

- Basic Control Flow:

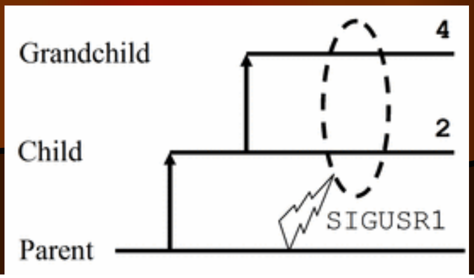


- Output is unpredictable because of two race conditions
  - Cannot predict order the child and grandchild will be scheduled for execution.
  - Cannot predict when they can receive the SIGUSR1 signal.
- Side Note: The picture is wrong, the grandchild exits with status 2, and the child with status 4.

Let's start with the basics.

3	Receives Signal Before First Child Forks
24	Children exit before signal sent.
42	Children exit before signal sent. (note unpredictability between the second child and first in scheduling)
23	One child finishes before the other receives a signal.
43	One child finishes before the other receives a signal.
33	Child forks grandchild, but parent sends kill before they exit.

- Is that all?



What about Exit()?

- Can't the children receive the signal after the printf but before the exit?

```
void Exit(int val)
{
  printf("%d", val);
  exit(0);
}
```

2433	Both child & grandchild printf before receiving sigusr1
4233	Same as 2433
243	One child exits and the other receives the sigusr1 signal after the printf.
423	Same as 243
233	One child finishes printf before receiving sigusr1.
433	Same as 233

Disclaimer: "Same as" implies reversing roles of child and grandchild.

Are we missing any?

- Some highly unlikely results since the unix specification does not guarantee timely receipt of signals.
- Unix based systems probably update all jobs during the same loop of the queue.
  - Disclaimer: Unless you're on a multi-processor machine! Then it is more likely to receive the signal at different times.

32	Child receives sigusr1 before grandparent which exits normally.	Here child receives sigusr1 before printf and grandchild.	323
34	Same as 32	Same as 323	343
234	Grandchild receives signal after printf, but child exits cleanly.	Here both receive signals after printf, but child after gchild.	2343
432	Same as 234.	Same as 2343	4323

Whoa… Did we get them all?

- Not exactly.
- Many system calls can fail.
  - Make note we take points off for any system calls in the shell lab if you ignore failure.
- Are you telling me that exit()/kill() can fail?
  - Of course not. They don't even have return values.
  - But kill may not do anything without correct permissions or if the provided process does not exist.
- Setpgid()?
  - If permission and the process exists, it should run deterministically.
- And fork()?
  - Absolutley. Eventually there will no longer be enough resource available to allocate to a new process.
  - This yields to two new potential cases one of blank output, and one of only the child outputting (i.e. "", or "4")

The Shell Lab - Tips

- Dog is man’s best friend.
- Man is your best friend.
- In a unix shell try:
  - man fork
  - man exit (oops that’s not the right one is it?)
  - man 2 exit (view the second entry for exit)
- This is a powerful tool that will help tremendously in the next three labs.
- Use it wisely; Use it all the time.

Provided Framework

- Main() sets up shell by initializing the job queue and waiting for shell input.
- After a line of text is received it calls eval().
- Provided eval() code calls the line parser and lets you focus on how to do the fork()ing and exec()ing.
- Make use of the provided job framework to make your life easier.
- Keep an eye, ear, and foot out for race conditions. They're tricky to find.

I'm stuck!?

- Read the handout.
- Read the provided code.
- Print your code and circle things you don't understand.
- Query your best friend. (i.e. man)
- Wipe the dust off your Systems Programming book.
- Run your shell interactively before running traces.
- Seek peers for general unix help.
- Test Hypotheses.
- Try the autolab message boards (I'd bet someone else has had the same question!)
- In final desperate times, search out your ~~least~~ favorite TA.

Sources

- Shell Lab (did you start?)
- The 213 Gremlin
  - Author: Tudor Dumitras
  - http://www.ece.cmu.edu/~tdumitra/gremlin/213_gremlin.htm
  - Note there are minor errors on the webpage. I hope that all of them have been fixed in these slides.
- Your Intro to Systems text book.
- Lastly, thank Nate for the slides!