CSCI 4061 Discussion 3

2/5/18

Recitation Grading Policy

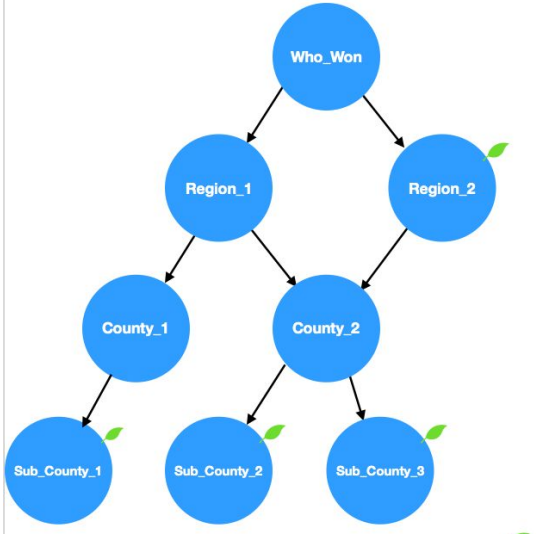
- Sign-in sheet must be signed for the 70% portion.
- The programming assignment **MUST** be submitted to Canvas by this Wednesday. Graded on effort.
  - Submit **ONLY** .c file(s).
  - Failure to sign-in, or submit means 0/70.
- The quiz is also due at the same time and is worth the remaining 30% of the grade for each recitation.
- This will be the policy for **all recitations going forward** unless amended.

Overview

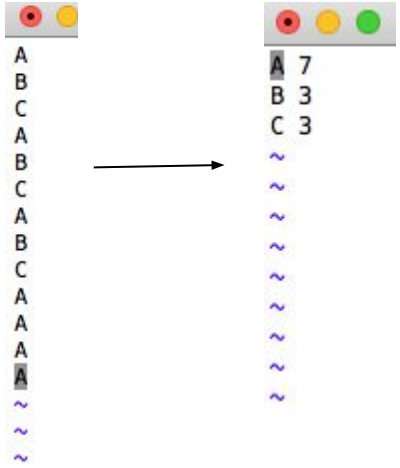
- Programming Assignment
- fork(), exec(), and wait()

Programming Assignment 1

- It is a Vote Counting Application
- For Voting, every area is divided into multiple sub-regions, which may be divided into multiple more.
- If you see closely, this forms a graph like structure.
- All leaf nodes shown will have a Votes File, containing individual votes

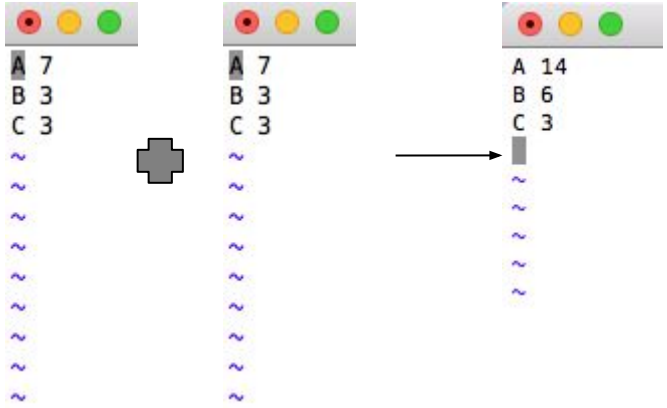


- The number and name of candidates are in a separate input file called input.txt
- The information to build a DAG is also in Input.txt
- You are supposed to build the DAG, then do Data aggregations.



- If it's a leaf node, it has to aggregate data as shown in the Figure above.
- If it's a non-leaf node, it has to aggregate data as shown in the next slide.

- The order of execution of processes matters here, otherwise the number of votes will differ.
- Thus, you must ensure only
  The processes that are independent of each other run simultaneously
- We have given two Test Cases. We may be testing on yet another set of Test Cases. You need to write your own makefile. You may place the executable file you obtain via ‘make’ into the TestCase folder and run it with input.txt



Programming Assignment 1

- Due Feb 21st, 11:59 PM.
- Only 1 person from your group has to turn it in, but everyone's name needs to be in the README
- Don't submit an executable, submit your code.

Example fork() Call

```
pid_t pid = fork();
if (pid > 0) {
   printf("I'm the parent!");
} else {
   printf("I'm the child!");
}
// Failure omitted.
```

Notes on fork()

- Both processes continue execution **at the very next line** after fork().
- OS gives **no guarantees** for which process will start running first.
- Stack/heap replicated, files/sockets are **NOT**.

Notes on the exec() call

- If successful, will **NOT** return to old program.

Example

```
// The char* cast is needed for NULL.
execl("...path...", (char*) NULL);

printf("The exec call failed!");
exit(1);
```

Variants of exec()

- The exec call has several variants, whose names end with l, v, p, and e.

Exec Contains Character ...	Meaning
l	Consumes list of args of constant size. (ends with char* NULL)
v	Consumes array of args of variable size.
p	Consumes filename instead of path. Uses default OS ‘path’ to find file.
e	Overrides default environment (another way to pass args).

Assignment

- Create a program that consumes an integer and spawns two child processes.
- Each child should make an exec call on the compiled program given, passing the int to rtime.o
  - One child will perform execl()
  - The other will perform execv()
- The compiled program given takes in an integer, **n**, and prints **2n** UNIX timestamps.
- Your program will be judged correct if all the timestamps are **unique**.