Grunnatriði stýrikerfa - dæmatímaverkefni 2

English follows.

Afurðin

- Verkefnið er að skrifa forrit í c sem keyrir á skipanalínu í Linux og gerir eftirfarandi:
- Tekur inn path á forrit sem parameter á skipanalínunni
- Tekur einnig inn þá parametra sem á að keyra forritið með í röð á eftir forritinu
- Býr til nýjan process sem keyrir forritið sem kom inn í parameter ásamt parameterunum inn í það forrit.
- Passar að það sem keyrða forritið skrifar út á skjá fari í staðinn gegnum pípu inn í yfirforritið
- Skrifar út það sem undirforritið skrifar út, en með forskeyti fyrir framan hverja línu.
- Bíður eftir að undirforritið ljúki keyrslu og lýkur síðan keyrslu sjálft.

Að sækja og skila verkefni

Til að hefjast handa, skráið ykkur inn á skel. Ef þið eruð á windows vél sækið putty og notið það til að skrá ykkur inn, annars getið þið notað SSH beint á skipanalínu. Slóðin sem þið tengist er notandanafn@skel.ru.is og lykilorðið er skólalykilorðið ykkar sem þið notið á myschool. Upptakan af fyrsta fyrirlestri sýnir ágætlega hvernig þið skráið ykkur inn á skel gegnum SSH með putty.

Þegar þið eruð komin inn á heimasvæðið ykkar ( ) getið þið keyrt eftirfarandi skipun:

tar -xvf /labs/usty17/lab2.tar

Núna ætti mappan /usty17_lab2 að hafa myndast í möppunni ykkar. Staðfestið það með skipuninni 'ls' og síðan:

cd usty17_lab2

Inni í þessari möppu á að vera '**Makefile**' og forritið **dirlstsample** sem þið getið notað sem prufuinntak á ykkar forriti. Í þessari möppu búið þið síðan til forritstextann og forritið ykkar. Þegar þið eruð tilbúin að skila getið þið keyrt eftirfarandi skipun:

make handin

Þessi skipun pakkar öllu innihaldi möppunnar saman og sendir inn í skilakerfið.

Vinnufyrirkomulag

Búið til skrána **processlab.c**, t.d. með '**vim processlab.c**'.

Útbúið c forrit með main() falli í skránni og þýðið það með:

gcc -o process processlab.c

Þetta mun búa til forritið **process** sem verður skilaforritið ykkar.

Útbúið test gögn og finnið absolute slóðir á þau forrit sem þið viljið testa ykkar forrit með, til þess að ganga úr skugga um að allt virki skv. lýsingu þegar verkefnið verður tilbúið.

Látið nú forritið gera eftirfarandi:

- Skrifa út mismunandi línur og mögulega keyra kóða sem tekur langan tíma, t.d. forlúppu sem keyrir 100000000 endurtekningar.
- Á undan þeim kóða, keyrið fallið fork(). Fork() skilar 0 í child process en skilar process id fyrir child processinn í parent processnum. Skoðið forritunarsessionið í lok fyrirlestrar 3 til að sjá þetta gert.
- Í parent processnum (ef pid != 0) látið forritið bíða með wait() skipun eftir því að child process ljúki keyrslu.
- Prófið að bæta við fleiri fork() köllum.
- Prófið að setja línurnar sem eru skrifaðar út og allt sem forritið gerir beint á eftir fork() og bíðið svo. Prófið einnig að bíða strax á eftir fork() og keyra keyrslurnar svo. Áttið ykkur á því sem er að gerast til að ná góðum skilningi á því hvernig fork() virkar.
- Keyrið execl eða execv til að skipta út forritinu í prócessnum fyrir annað forrit, t.d. ./dirlst forritið. Athugið að seinasta færslan í lista yfir parametra verður að vera (char*)NULL.
- Prófið núna að búa til pípu með pipe() fallinu. Skoðið fyrirlestur þrjú og gögn sem fylgja honum undir "annað" til að nota þetta rétt.
- Látið child process setja skrifendann á pípunni á standard outputtið sín megin.
- Skrifið eitthvað út með puts() í child processnum og lesið það úr lesendanum á pípunni í parent processum, t.d. með fgets (sjá skjal með fyrirlestri 3). Parent getur svo sjálfur skrifað þetta út aftur á sitt standard output, sem er ennþá bara skjárinn.
- Nú þegar farið er að lesa og skrifa með pípunni og það sem child skrifar á stdout er lesið inn í parent má einfaldlega láta child skipta sér út fyrir annað forrit með execl eða execv. Allt sem það forrit skrifar út á þá að fara um pípuna.
- Nú þarf að útbúa parent þannig að hann lesi úr pípunni línu fyrir línu og skrifi hverja línu út með forskeyti sem hann bætir sjálfur við, t.d. "This just in from child: ".
- Þegar ekkert fleira kemur úr pípunni (End of file (EOF) fundið eða fgets skilar NULL) má parent einfaldlega kalla á wait til að ganga frá child process og svo klára sitt rennsli (parent).

Hvenær sem þið eruð ekki viss um hvað skal gera næst, finnið leið til að skrifa út eitthvað sem gæti hjálpað ykkur, t.d. errno eða innihald breyta eða eitthvað þess háttar.

Lesið vel lýsingar á öllum föllum sem þið notið, t.d. á man síðum á netinu eða á **pubs.opengroup.org**. Þar kemur fram hvað föllin gera, t.d. ef þau skila einhverju og hækka e-n tekljara á sama tíma, eða ef þau skila ’-1’ eða NULL og setja gildi í **errno** á sama tíma. Geriði ykkar besta til að fylgja þessum vísbendingum og klára verkefnið á þann hátt.

Góða skemmtun!

The Product

The project is to write a program in c that runs in the linux terminal and does the following:

- Takes a path to a executeble program as a parameter
- Also takes the parameters that the executable program should be run with as further parameters
- Makes a new process that runs the program that was taken in as a parameter with the parameters that followed.
- Makes sure that what the run program writes to the standard output goes through a pipe into the main program instead.
- Writes what the "inner" program sends through the pipe, line by line, but adding a prefix on every line.
- Waits for the inner program to finish and then finishes itself.

Getting and returning the assignment

To begin, log into skel. On windows use putty or use ssh on linux to connect directly to skel. The path is username@skel.ru.is and the password is you school AD password that you use for myschool and mail. The recording of the first lecture shows nicely how to log onto skel and make, compile and run your first c program.

Once in your home folder you can do the following:

tar -xvf /labs/usty17/lab2.tar

Now the folder /usty17_lab2 should be in your home directory. Make sure by typing 'ls' and then enter it with:

cd usty17_lab2

In this folder there should be a **Makefile** and the program **dirlstsample** that you can use as test input into your program. In this folder you will make your program. Once you are ready you can enter the following:

make handin

This archives the entire contents of the folder and returns it.

Process

Make the file processlab.c, for example with 'vim processlab.c'.

Make a c program with a main() function in the file and compile it with:

gcc -o process processlab.c

This will make the program process that that will be your handed in program:

Make test documents and find absolute paths to the programs that you will test your program on, to make sure everything works as described one it's ready.

Now make your program do the following:

- Print out some text and possibly run code that takes a while (a few tenths of a second), like a for loop that runs 100000000 cycles of nothing.
- Before that code, run the function fork(). Fork() returns 0 in the child process but returns the process id of the child in the parent process. Look at the programming session at the end of lecture 3 to see this done.
- In the parent process (if pid != 0) make the program wait for the child process to finish, using the wait() command.
- Try adding more calls to fork().
- Try putting the code that writes and does stuff directly after fork() and then do wait(). Also try to wait right after fork() and do stuff after. Try to interleave this. Get a feel for what is going on and how fork() and wait() work.
- Run execl or execv to switch the program in the process for another program, for example the ./dirlst program. Make sure the last parameter in the list of parameters must be (char *)NULL.
- Now try making a pipe with the pipe() function. Look at lecture 3 and the .pdf (in english) that comes with that lecture on myschool to use this correctly. Alternatively read about the usage on man pages or the open group.
- Make the child process set the write-end of the pipe to its standard output.
- Write somthing with puts() in the child process and read it from the read-end of the pipe in the parent process, for example with fgets (see document with lecture 3). The Parent can then write it out into its own standard output, which is still the screen.
- Now that the program is writing and reading from the pipe, and everything the child writes into its standard output is read by the parent, you can simple switch out the child process for a different program using execl or execv. Everything that program prints to the "screen" goes into the pipe instead.
- Now you have to program the parent so that it reads from the pipe, line by line, and writes each line out with a prefix, for example "This just in from child: ".
- When nothing more comes from the pipe (End of file (EOF) found or fgets returns NULL) the parent can simply call wait to make sure the child process is cleaned up and then finish its own run (parent).

Whenever you're not sure what to do next, find a way to print something to the screen that might help you, for example errno or the contents of variables.

Read well the descriptions of all functions you use, for instance on the man pages (in linux or online) or at pubs.opengroup.org. There you can see what the functions do, for instance if the return a value or raise some counter at the same time or if the return '-1' or NULL and set the value of errno at the same time. Follow those leads and finish this assignment through exploring.

Have a good time!