Operating Systems HW #3

Due 23.05.2016, 23:55

In this exercise, we develop a character counter as a multiprocessing system. Given a text file, our goal is to find out how many instances of a given character are in that file. We create two programs to do that: Dispatcher and Counter.

Dispatcher (dispatcher.c)
This program launches several Counter processes, telling each process which part of the text file the process should work on. Dispatcher waits for all child processes to finish, aggregating their results. Finally, Dispatcher prints out the total result.

Command line arguments:
argv[1] – the character to count,
argv[2] – the text file to process.

The flow:
1. Determine the file size – *N.* Decide *K* – the size of the chunk each Counter processes, and *Q* – the number of the Counter processes. Observe that $N = K*Q$. If *N* is less that twice the memory page size, then let *Q* be 1. Constrain the decision by $Q \leq 16$.
2. Register USR1 signal handler. Turn on SA_SIGINFO flag, so your handler function gets more arguments. See Appendix A for an example.
3. Launch *Q* Counter processes. Use fork() and execv() pair, as shown in the class.
4. Wait for all processes to finish.

Upon USR1 signal:
1. Determine the signal sender PID. See Appendix A for an example.
2. Open a named pipe file called “/tmp/counter_*PID*” for reading, where the *PID* suffix is the PID above.
3. Read the current Counter result from the pipe.
4. Aggregate the result in some global data structure.

Counter (counter.c)
This program maps the relevant part of the text file into the virtual address space. The program iterates through the mapped array and counts the requested character. It creates a named pipe to report result to Dispatcher. It sends USR1 signal to Dispatcher. It writes the result to the named pipe.

Command line arguments:
argv[1] – the character to count,
argv[2] – the text file to process,
argv[3] – the offset in the text file to start counting from,
argv[4] – the length of the relevant chunk to process.

The flow:
1. Map the relevant chunk of the text file into the memory as an array.
2. Iterate through the array, count the characters. Obtain result *R*.
3. Determine own process ID – *PID.*
4. Create a named pipe file with the name "/tmp/counter_*PID*". Open it for writing.
5. Determine Dispatcher process ID. Use getppid() function, since Dispatcher is the parent process.
6. Send USR1 signal to the Dispatcher process.
7. Write *R* into the pipe.
8. Sleep for 1 second.
9. Unmap the file, close the pipe, delete the pipe file. Exit.

Remarks
1. There are other ways to communicate between Counters and Dispatcher while reporting results. You can propose an alternative solution. Email your proposal to evgenyl@post.tau.ac.il and we consider it. No solutions involving temporally regular files will be approved.

2. You submit "ex3_012345678.zip" file. Where "012345678" is your ID (תעודת זהות) number. Two source files – `dispatcher.c` and `counter.c` are in this archive.

2. You can check the correctness of you program issuing the following command (just substitute "a" by the character of your own, and take the relevant file).

```
$ awk 'BEGIN{FS="a"; R=0;}{R=R+NF-1;}END{print R;}' aaa.txt
```

The command above wasn't tested extensively. Report a bug, if found.

Appendix A

The following code shows how to determine sender process ID while handling a signal.

```c
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>



//-----------------------------------------------------
void my_signal_handler( int         signum,
                        siginfo_t*  info,
                        void*       ptr)
{
  printf("Signal sent from process %lu\n",
         (unsigned long) info->si_pid);
}
//-----------------------------------------------------
int main()
{
  // Structure to pass to the registration syscall
  struct sigaction new_action;
  memset(&new_action, 0, sizeof(new_action));

  // Assign pointer to our handler function
  new_action.sa_handler = my_signal_handler;

  // Setup the flags
  new_action.sa_flags = SA_SIGINFO;

  // Register the handler
  if( 0 != sigaction(SIGUSR1, &new_action, NULL) )
  {
    printf("Signal handle registration "
           "failed. %s\n", strerror(errno));
    return -1;
  }

  while( 1 )
  {
    sleep( 1 );
    printf("Meditating\n");
  }
  return 0;
}
//================== END OF FILE =====================
```