timekeeper – a new system utility program

PROGRAMMING PROJECT

Target

- You are going to implement a C program (or C++ program) which reports to the end-users on the execution statistics and termination status of a program or a sequence of programs connected by pipes

Objectives

- An assessment task related to ILO4 [Practicability]
- A learning activity related to ILO 2a
- To learn how to use various important Unix system functions
  - to perform process creation and program execution;
  - to interact between processes by using signals and pipes; and
  - to get process's running statistics.

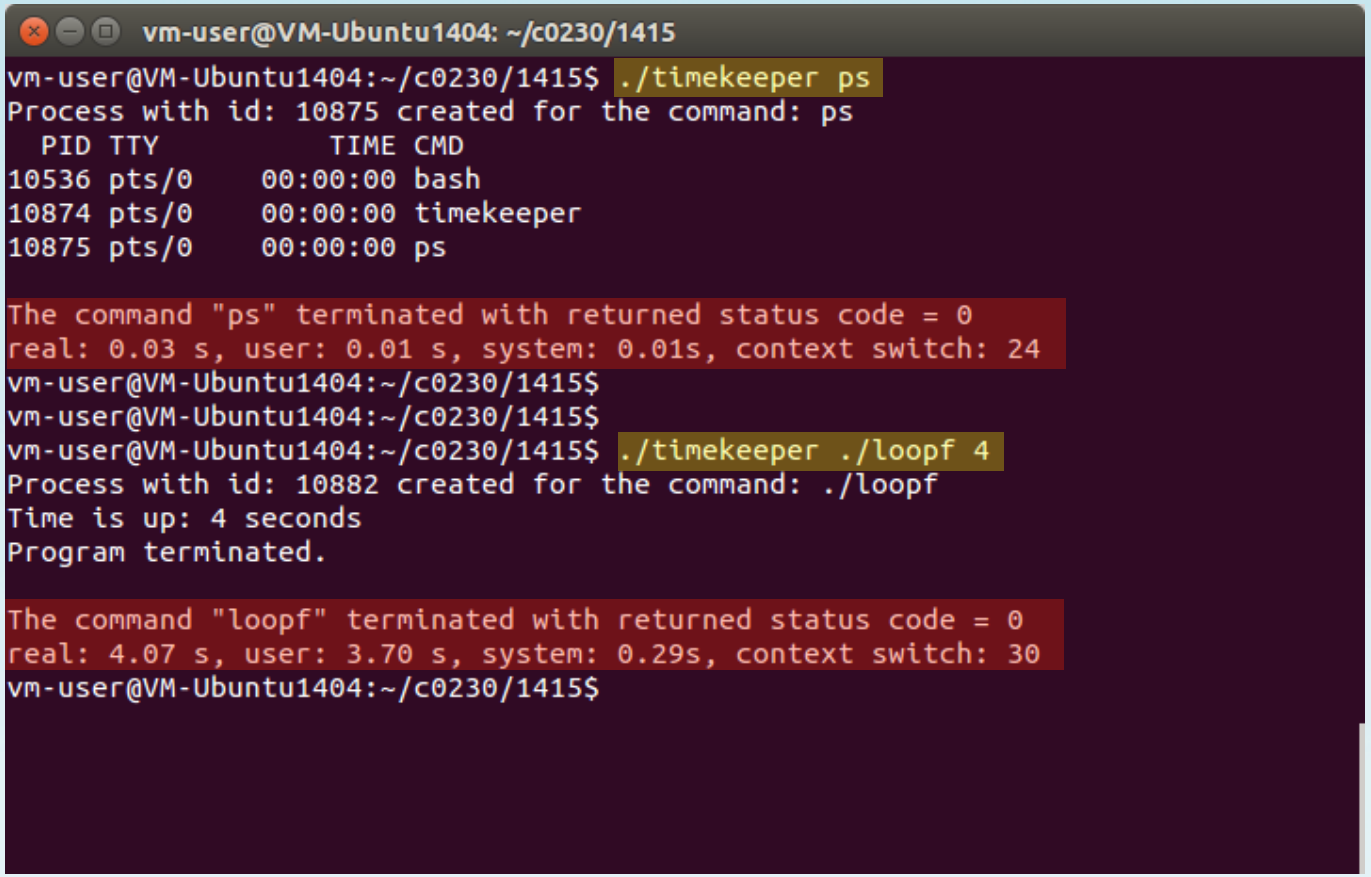
Features

- ***timekeeper*** accepts command-line input arguments
- ***timekeeper*** interprets and parses input arguments to form commands (with associated arguments)
- ***timekeeper*** starts all commands as child processes and waits for their termination
- When detecting a child process has terminated, ***timekeeper*** will print out termination status and running statistics of that terminated child process

Features

Output from *ps*

Output from *loopf*



Features

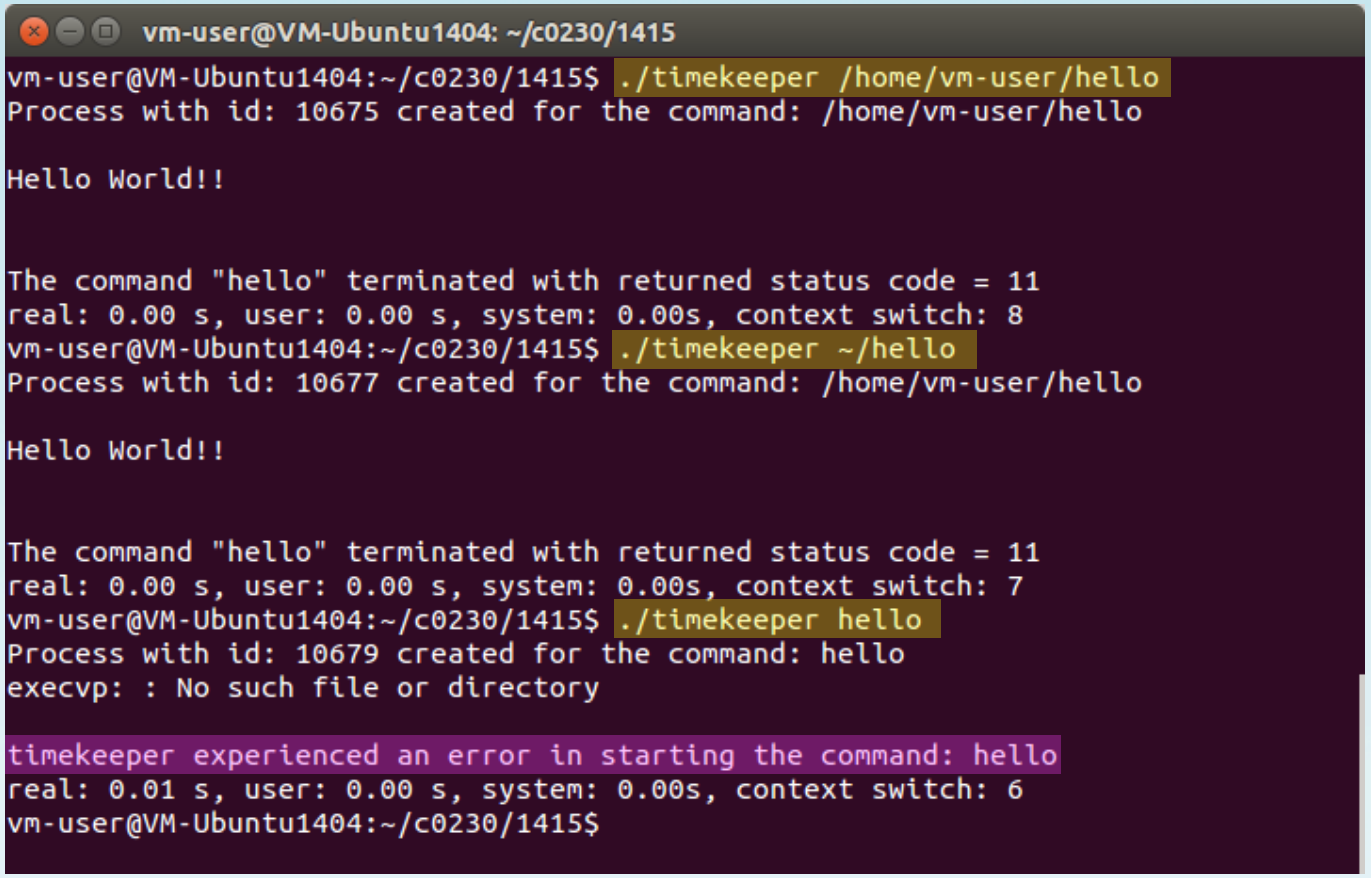
- ***timekeeper*** is able to locate and execute any valid program by giving an absolute path (starting with /) or a relative path (starting with ./) or by searching directories under the $PATH environment variable
- If the command cannot be found or executed, ***timekeeper*** displays an error message

Features

Absolute path

Another way to indicate the absolute path

No explicit path info



Cannot start the command

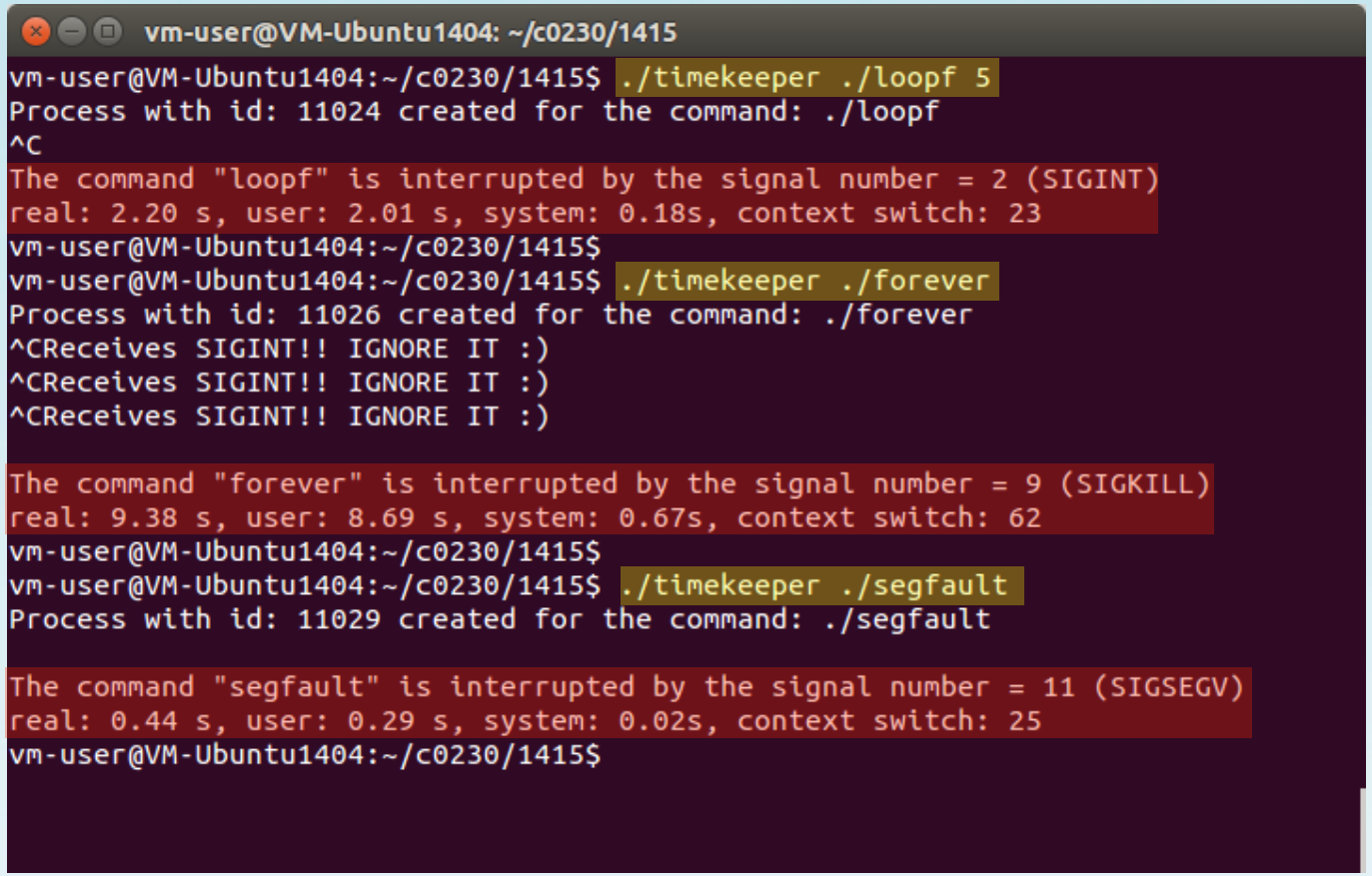
Features

- Upon receiving SIGINT signal, ***timekeeper*** will not be affected by this signal and will continue until it detects that all its child processes have terminated
- Upon receiving SIGINT signal, the child process(es) should response to the signal as according to the predefined behavior of each command in response to the SIGINT signal
- ***timekeeper*** outputs a message to indicate that the child process is terminated by a specific signal

Features

^C is pressed

Output from *forever*; it cannot be terminated by ^C



loopf terminated involuntary by ^C

forever terminated involuntary by signal SIGKILL

segfault terminated involuntary as it encountered segmentation fault

Features

- Coordination between ***timekeeper*** and its child process(es)
  - All child process(es) wait(s) for SIGUSR1 signal (sent by ***timekeeper***) before start executing the target command

Features

- A sequence of programs connected by pipe operators
  - ***timekeeper*** uses a special symbol '**!**' to act as the pipe operator
  - cannot use tradition pipe symbol '|' as this would confuse the shell process
  - these programs will be linked in a logical pipe as according to the sequence of programs
  - ***timekeeper*** prints out the termination status (or signal information) and running statistics of each terminated child process
  - Upon detecting all child processes have terminated, ***timekeeper*** will terminate too

Features

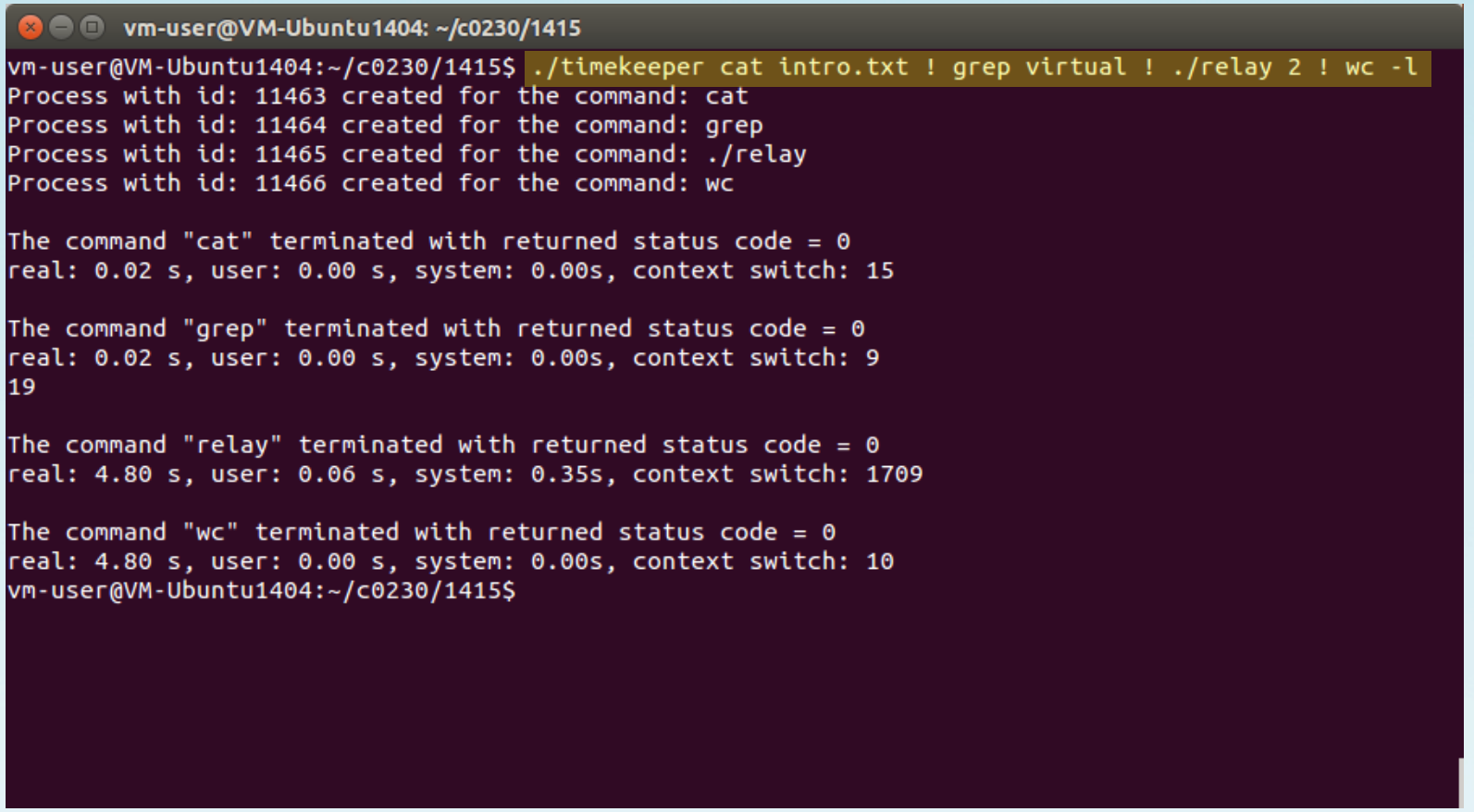
Output from *wc*

Output from *wc*

```
vm-user@VM-Ubuntu1404: ~/c0230/1415
vm-user@VM-Ubuntu1404:~/c0230/1415$ ./timekeeper cat timekeeper.c ! wc -l
Process with id: 11434 created for the command: cat
Process with id: 11435 created for the command: wc
233

The command "cat" terminated with returned status code = 0
real: 0.01 s, user: 0.00 s, system: 0.00s, context switch: 11

The command "wc" terminated with returned status code = 0
real: 0.01 s, user: 0.00 s, system: 0.00s, context switch: 11
vm-user@VM-Ubuntu1404:~/c0230/1415$
vm-user@VM-Ubuntu1404:~/c0230/1415$ ./timekeeper cat intro.txt ! grep virtual ! wc -l
Process with id: 11437 created for the command: cat
Process with id: 11438 created for the command: grep
Process with id: 11439 created for the command: wc

The command "cat" terminated with returned status code = 0
real: 0.01 s, user: 0.00 s, system: 0.00s, context switch: 14
19

The command "grep" terminated with returned status code = 0
real: 0.01 s, user: 0.00 s, system: 0.00s, context switch: 6

The command "wc" terminated with returned status code = 0
real: 0.00 s, user: 0.00 s, system: 0.00s, context switch: 6
vm-user@VM-Ubuntu1404:~/c0230/1415$
```

Features

Output from *wc*



Subdivide the task into Four stages

- Lab 1 – Sept 22 (Mon) at HW312
  - learn how to create a new process to execute a valid command
- Lab 2 – Sept 29 (Mon) at HW312
  - learn how to install a signal handler to handle a specific signal
- Lab 3 – Oct 6 (Mon) at HW312
  - learn how to read and use those process's statistics
- Lab 4 – Oct 20 (Mon) at HW312
  - learn how to use pipe() system functions to set up a pipe between two processes

4 Lab sessions

- Lab Exercise specification will be issued 24 hours prior to the Lab Session, and you are encouraged to read it beforehand
- During the Lab Session, there will be some tips and help going to be offered to you
- As long as you attend the Lab sessions, you should find that the mechanisms are quite straightforward and simple to use. In addition, the Lab exercise programs are easy and straightforward.
  - Ernest and I would always be glad to help you out if you got any problem with the course materials.

Submissions

- Lab 1 [Optional]
  - Deadline – 23:59pm, Sep 26, 2014 (Friday)
  - Name file as: uid_username_timekeeper_lab1.c
- Lab 2 [Optional]
  - Deadline - 23:59pm, Oct 3, 2014 (Friday)
  - Name file as: uid_username_timekeeper_lab2.c
- Lab 3 [Optional]
  - Deadline - 23:59pm, Oct 10, 2014 (Friday)
  - Name file as: uid_username_timekeeper_lab3.c
- Final [Compulsory]
  - Deadline - 23:59pm, Oct 31, 2014 (Friday)
  - Name file as: uid_username_timekeeper.c

Submissions

- Lab 1, Lab 2, & Lab 3 submissions are optional but highly recommended
  - Will accept submissions even after the deadlines but will be labelled as *late* (with no penalty)
- Final submission – Compulsory & LATE SUBMISSION WILL NOT BE ACCEPTED
  - Submission systems for Lab 1, Lab 2, Lab 3, & Final will be closed by the Final Deadline

Grading Criteria

- Please read project specification document

Documentation (1 point)	High Quality (0.5 point)
	Standard Quality (0.5 point)
Correctness of the program (13 points)	Lab 1 Exercise (2 points) – Process creation
	Lab 2 Exercise (2 points) – Signal handling
	Lab 3 Exercise (2 points) – Report timing
	Final program (7 points)

Grading Policy

- The tutor will first test your final submission after the project deadline
- If your final program fully complies with the project specification, you'll get all the marks for all four stages automatically.
- Otherwise, tutor will test your Lab3 submission (if any). If it passes all test cases, you'll get all the marks for the first three stages.
- Otherwise, tutor will test your Lab2 submission (if any). If it passes all test cases, you'll get all the marks for the first two stages.
- Otherwise, tutor will test your Lab1 submission (if any) to give marks.
- If you want to get stage marks earlier, please arrange a demo session with tutor to test your program, during the consultation hours or by appointment.

Computer platform to use

- You are expected to work on any Linux systems (preferred on the latest distribution of Fedora or Ubuntu)
- Your programs are written in C (or C++) and successfully compiled with gcc or g++
- Your submission(s) will be primarily tested under Ubuntu 14 (installed in HW311/312). Make sure that your program can be compiled *without any error or warning*. Otherwise, we have no way to test your submission(s) and thus you may lose *all* the marks