

Relazione del progetto “Top”

Ingegneria Informatica, Sistemi operativi, a.a. 2021-2022

Lorenzo Pecorari, matricola 1885161,

pecorari.1885161@studenti.uniroma1.it

Cosa prevede

L'obiettivo prefissato dal codice è quello di rappresentare un programma in linguaggio C che tenga sotto controllo l'insieme dei processi presenti nel sistema eseguito dalla macchina. Le principali funzioni che sono disponibili permettono di terminare, uccidere, sospendere o riprendere tali processi, accedendo alle informazioni necessarie mediante */proc* del filesystem. Altro aspetto messo in risalto dal programma riguarda il monitoraggio delle risorse utilizzate da ogni singolo processo, quali il carico sulla CPU e la memoria occupata. Infine, allo scopo di permettere un utilizzo più agevole del software, sono state implementate funzionalità di ordinamento, stampa a schermo delle informazioni e ricerca dei processi stessi.

Come funziona

Tale software, appoggiandosi alle funzioni dello standard POSIX e ad altre create ad hoc, effettua una scansione della sopracitata directory */proc* per ottenere informazioni sui vari processi in esecuzione nel sistema, gestirli e stampare a schermo il risultato finale.

La funzione da cui tutto il programma ha inizio è *program_runner*, la quale ha come argomenti un puntatore di tipo *DIR** e una struttura *dirent**.

Lettura di */proc* e ottenimento dati

Partendo da tali variabili di tipo *dirent** e *DIR**, ciò che è ricavato da ogni

```
typedef struct proc{  
    int pid;  
    char name[NAME_SIZE];  
    char cmdline[BUF_SIZE];  
    char status;  
    long long unsigned starttime;  
    long unsigned utime;  
    long unsigned stime;  
    long unsigned children_time;  
    long unsigned tot_time;  
    long unsigned mem_usage;  
    double load_percentage;  
} proc;
```

lettura di tale cartella viene memorizzato in una struttura appositamente creata e che rappresenta ogni singolo processo “trovato”, descrivendolo mediante un insieme di campi contenenti le informazioni necessarie al corretto funzionamento del programma. I processi salvati e rappresentati con tale struttura sono contenuti in un dati array di tipo *proc*, avente un *MAX_PROCESS*

Struttura che contiene le informazioni di un processo

elementi e definito come *procs*; sarà questo ad essere acceduta durante il corso dell'esecuzione del programma.

Una volta aperta */proc*, utilizzata *opendir* ed effettuati i relativi controlli, viene impiegata ciclicamente *readdir* per scandire tutti gli elementi di tipo directory presenti (*/proc/[pid]*) finché non sarà selezionata la funzione di uscita. Dell'inserimento dei processi all'interno di *procs* si occupa *insert_process*, che a sua volta utilizza *get_stats*, funzione che richiama *get_cmdline* e *get_stat* per rilevare i dati necessari. Seguendo l'ordine, *get_cmdline* sfrutta il file *cmdline* per ottenere la riga di comando completa del processo mentre *get_stats* usufruisce del ben più complesso *stat*. Attraverso una *read* viene portata avanti la scansione di *stat*, accedendo ad alcuni dei diversi campi. Di fatto vengono estrapolate informazioni solo dai numeri 2, 3, 14, 15, 16, 17, 22, 24; questi corrispondono rispettivamente al nome e allo stato del processo, al tempo in *user* e *kernel mode*, al tempo atteso per la schedulazione dei processi figli in *user mode*, al tempo di avvio dopo il boot del sistema e al numero di pagine che il processo ha nella memoria fisica.

Stampa a schermo

Al fine di rendere disponibile graficamente il risultato delle precedenti funzioni, di base viene fatto uso della stampa su terminale mediante le syscall *printf* e *write*. Le informazioni ottenute precedentemente sono rappresentate in forma tabellare per ciascun processo, mediante *print_processes*, la quale stampa delle informazioni di carattere generale quali il numero di processi, il valore di *uptime* e il carico della CPU. Tale funzione si appoggia però a *print_table*, funzione accedente a *procs* per ricavare le informazioni necessarie e riprodurre graficamente i campi principali della struttura con cui viene rappresentato ogni processo. Di default mostra in ordine decrescente i 10 processi con maggior consumo della CPU; è però modificabile dall'utente nel corso dell'esecuzione del programma, sia il numero delle righe che il tipo di ordinamento.

L'aggiornamento delle informazioni avviene periodicamente ogni secondo, prevedendo l'ottenimento di nuove informazioni, effettuando un refresh del terminale mediante la macro *clrscr* e la stampa delle nuove informazioni attraverso le funzioni prima citate. La *clrscr* fa uso della sequenza di caratteri usata dalla funzione *clear* comunemente utilizzata per "pulire" il terminale, permettendo di avere sempre allineati i nuovi dati.

PID	NAME	PATH	STATUS	TIME	MEMORY (rss)	CPU LOAD
1533	systemd	/lib/systemd/systemd	S	84841	2665	1.41 %
1631	Xorg	/usr/lib/xorg/Xorg	S	26481	26528	0.44 %
1766	gnome-shell	/usr/bin/gnome-shell	S	18216	71646	0.30 %
2297	soffice.bin	/usr/lib/libreoffic...	S	17292	68503	0.29 %
1539	pulseaudio	/usr/bin/pulseaudio	S	7629	5407	0.13 %
2577	gedit	/usr/bin/gedit	S	6148	31459	0.10 %
227	irq/57-EL...	/sbin/init	S	2961	0	0.05 %
5204	kworker/5...	bash	I	1054	0	0.02 %
6731	chrome	/opt/google/chrome/...	.	954	4006	0.02 %
1866	nautilus	/usr/bin/nautilus	S	913	25049	0.01 %

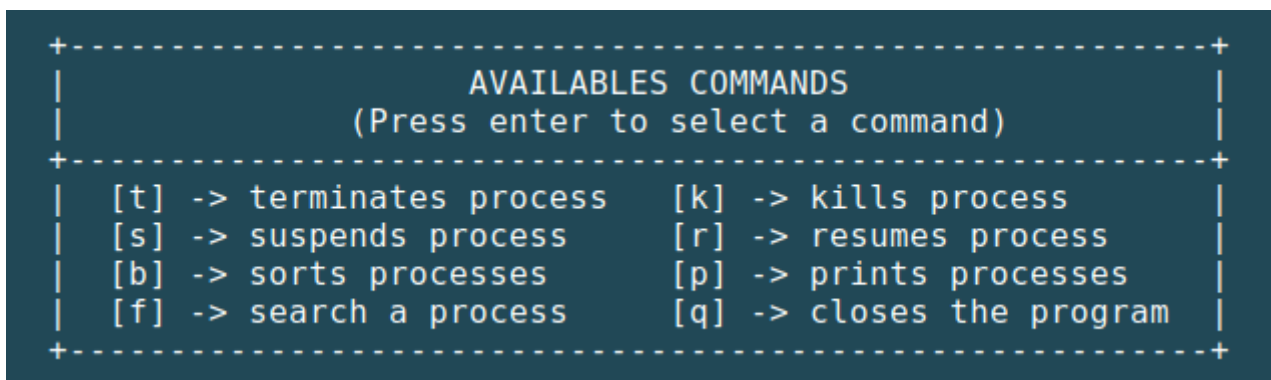
Tabella di default stampata dal programma

Interazione con l'utente

Per interagire con l'utente, è previsto l'uso di *act*, una struttura *sigaction* predisposta a rilevare segnali di tipo *SIGALRM* e a reagire mediante un apposito handler, e un *thr*, un thread secondario e parallelo a quello principale. Lo scopo di *thr*, gestito da *thread_handler*, è usare una pipe su *stdin* per rilevare, mediante un ciclo *while* e una *read*, se viene premuto il tasto invio, carattere corrisponde all'intero 10, e memorizzarlo in una variabile ausiliaria comune con il thread principale. Prima di aprire tale pipe, che prevede anche il lancio di *sleep* di un secondo al fine di permettere la rilevazione del carattere *\n* e mandare il programma principale in sospensione attraverso una *pause*, viene richiesto l'invio di *SIGALRM*. Una volta che tale segnale viene catturato, si attiva *sigalarm_handler*, il gestore della struttura *act*, la quale verifica sulla variabile destinata a registrare *line feed* sia stato effettivamente memorizzato qualcosa. Nel caso dovesse verificarsi tale corrispondenza, il programma attende che l'utente inserisca un carattere attraverso la funzione *choose_command*. Se il comando risulta valido e corrispondente ad una delle funzioni previste allora avviene l'attivazione della corrispondente altrimenti viene ripreso l'esecuzione di default, sospesa finché l'utente non inserisce un comando e/o non viene portata a termine la schedulazione della funzionalità selezionata.

Funzioni disponibili

Il riconoscimento della funzionalità scelta avviene mediante *choose_command*, la quale cerca di riconoscere uno dei casi possibili e segnala il comando selezionato per utilizzare *command_runner*, specializzata nel lanciare la funzione idonea alla richiesta.



```
+-----+
|                                     |
|             AVAILABLES COMMANDS   |
|      (Press enter to select a command) |
|-----+-----+
| [t] -> terminates process   [k] -> kills process   |
| [s] -> suspends process     [r] -> resumes process  |
| [b] -> sorts processes      [p] -> prints processes  |
| [f] -> search a process     [q] -> closes the program |
|-----+-----+
```

Tabella contenente i comandi disponibili; viene sempre mostrata all'utente

Se selezionata una tra le quattro funzionalità (*t*, *k*, *s*, *r*) di manipolazione dei processi, attraverso la funzione *get_proc_pid*, viene richiesto l'inserimento da tastiera del pid allo scopo di poter successivamente utilizzare la funzione *kill* della libreria *signal.h*. La ricerca del processo avviene secondo una scansione lineare di *procs*, al fine di verificarne la presenza o meno nel sistema del processo con pid richiesto. Nel caso negativo non viene lanciato nessun segnale e avviene una stampa d'errore per segnalarlo, altrimenti è inviato il segnale specifico al

processo desiderato. Viene inviato un segnale di SIGTERM nel caso in cui sia stata selezionata la terminazione (*t*), SIGKILL se il comando scelto è di uccidere il processo (*k*), SIGSTOP e SIGCONT per la sospensione (*s*) e la ripresa (*r*) rispettivamente. L'utente può fruire di alcune informazioni segnalate con la stampa a schermo una volta che la funzione kill ha terminato l'esecuzione.

Nel caso in cui venga selezionata la funzione per il sorting (*b*), si attiva *select_sorting* che dapprima richiede un parametro e un tipo secondo cui effettuare l'ordinamento, successivamente viene utilizzata *bubblesort* per sistemare i processi da stampare secondo la modalità desiderata. Per il numero di righe che la tabella rappresentate i processi deve avere, è possibile utilizzare la funzionalità (*p*) che, mediante la funzione *select_procs_to_print*, richiede all'utente di inserire in un range tra 0 e 50. Una volta selezionato un valore idoneo, il programma effettua un ridimensionamento della finestra attraverso le macro *resize_scr_small*, *resize_scr_medium* e *resize_scr_large* in base a quante righe l'utente desidera vedere nella tabella. Tali macro impiegano delle "ANSI escape sequences" che vengono interpretate dal terminale come sequenze per manipolare la finestra. Per dettagli si rimanda al seguente link <https://invisible-island.net/xterm/ctlseqs/ctlseqs.html>.

È possibile ottenere informazioni su un singolo processo facendo uso della funzione dedicata (*f*), la quale richiede la modalità ricerca (nome/pid), e il parametro di ricerca grazie a *find_process*. In appoggio a tale vi sono *get_process_from_name* e *get_process_from_pid* rispettivamente. Nel caso di una scansione per nome, potrebbe esservi la possibilità di più processi aventi lo stesso nome e quindi, per ovviare a tale evenienza, vengono inseriti tutti i casi all'interno di un array con elementi di tipo *proc*. Se si verifica una corrispondenza veritiera, viene riportato a schermo l'insieme dei processi con tale nome ed in particolare vengono specificati pid e comandi relativi ad essi. Invece, grazie all'univocità di cui gode il process identifier (riferendosi a sistemi Unix-like), la ricerca per pid, nel percorso positivo, il programma restituisce una piccola tabella con informazioni essenziali sul processo (quali pid, nome, comando e tempi), altrimenti appare a schermo una stringa di errore.

Come compilare ed eseguire

La compilazione del software è legata all'utilizzo di un makefile, è solamente necessario digitare *make* all'interno del terminale nella cartella del software per ottenere il file eseguibile. Per eliminare l'eseguibile (ed eventuali file con estensione .o) è possibile utilizzare il comando *make clean*.

Per l'esecuzione è necessario inserire il comando *./main* da riga di comando dal terminale.

Una volta lanciato il programma, dapprima ridimensiona la finestra del terminale per le dimensioni adeguate, di base utilizzerà *resize_scr_small*,

e successivamente inizia a stampare ciclicamente le informazioni sui processi in forma tabellare. Per accedere ad una funzionalità è necessario premere invio e attendere la richiesta di inserimento del comando. Se il comando è corretto il programma risponderà con le richieste per portare avanti l'esecuzione della funzione desiderata altrimenti riprenderà la normale esecuzione. Per chiudere il programma è sufficiente ripetere l'operazione di inserimento di un comando e digitare "q". Altri comandi, diversi da quelli previsti, vengono semplicemente ignorati dal programma, il quale torna alla stampa ciclica ad ogni carattere errato inserito. Tutte le funzionalità sono dotate di un modo per tornare indietro ed annullare la selezione del comando. Accanto alla richiesta di inserire un argomento/parametro, viene suggerito l'inserimento di 'esc' per annullare tale selezione.