Relazione HW1 - Realizzazione del comando finger

Nome: Rideewitage Lachitha Sangeeth Cognome: Perera

Matricola: 2042904

Introduzione

Il comando finger restitusce (in base all'opzione specficata) le informazioni degli utenti loggati o degli utenti/processi (anche non loggati) specificati come argomenti dall'utente che chiama il comando.
Opzioni comandi finger:

- **-s** : Tipo di formattazione delle informazioni da stampare secondo il seguente schema:
  Login Name Tty Idle Login Time Office Phone
- **-l** : Tipo di formattazione delle informazioni da stampare secondo il seguente schema:
  Login: nomeUtente RaleName: nomeReale
  Directory: /Users/nomeUtente Shell: /bin/tipoShell
  Login time + tty + host
  ultima mail arrivata e/o ultima volta che e' stata letta la mailbox
  contenuto dei file .plan .projects .pgpkey
- **-m**: Opzione che se indicata, e sono stati specificati altri argomenti, non esegue il controllo sul nome reale
- **-p**: Opzione che se indicata con l'opzione -l non stampa il contenuto dei file .plan .projects e .pgpkey

Per la realizzazione del comando sono state usate le librerie di c chiamate **utmp.h** e **pwd.h**. Attraverso la libreria utmp.h e' possibile ottenere gli utenti loggati, con la relativa data di login, la tipologia del tty, l'host e il nome utente; mentre dalla libreria pwd.h e' possibile ricavare tutti gli utenti e i processi di loggati e non nel proprio sistema, ricavando le informazioni delle GECOS (composta da: nome reale, room number, office phone, home phone), la directory dell'utente e della shel utilizzata dall'utente. Invece, per controllare la mail arrivate ricaviamo le informazioni a noi necessarie dal file situato in /var/mail/nomeUtente; mentre i file .plan .projects e .pgpkey sono nella directory principale dell'utente.

Struct che utilizzo

Per poter salvare le informazioni di ogni utente, utilizzo una strutc chiamata **user_inf** composta da:

1. user_name -> in cui viene salvato il nome utente (che posso ottenere sia in pwd.h che in utmp.h)
2. real_name -> in cui viene salvato il nome reale (ottenibile dalla libreria pwd.h all'interno del GECOS)
3. usr_dir -> in cui viene salvata la directory dell'utente (ottenibile dalla libreria pwd.h)
4. login_time -> in cui viene salvato quando l'utente ha effettuato l'accesso come una struttura tm* (ottenibile dalla libreria utmp.h)
5. shell_dir -> in cui viene salvata la directory della tipologia di shell utilizzata dall'utente (ottenibile dalla libreria pwd.h)
6. room_number -> in cui viene salvato il room number (ottenibile dalla GECOS dell'utente dalla libreria pwd.h)
7. office -> in cui viene salvato l'office phone (ottenibile dalla GECOS dell'utente attraverso la libreria pwd.h)
8. home -> in cui viene salvato l'home phone (ottenibile dalla GECOS dell'utente nella libreria pwd.h)
9. last_mail -> in cui viene salvato quando e' arrivata l'ultima mail (ottenibile dal file /var/mail/nomeUtente)
10. last_mail_read -> in cui viene salvato l'ultima volta che l'utente ha salvato la mail (ottenibile dal file /var/mail/nomeUtente)
11. timezone -> in cui viene salvato il fuso orario
12. tty -> in cui viene salvata la tipologia del dispositivo di comunicazione usato (esempio il tty o pts) (ottenibile dalla libreria utmp.h)
13. host -> in cui viene salvato l'host (ottenibile dalla libreria utmp.h)
14. idle_time -> in cui viene salvato da quanto tempo la tipologia del dispositivo di comunicazione usato dall'utente e' attivo (ottenibile attraverso la funzione stat dal file /dev/tipologiaDiTemirnale)
15. bool exist -> variabile booleana per indicare l'esistenza di un utente (utilizzata per quando salvo piu' utentu in un array e segnare fino a dove iterare)
16. bool logged -> variabile booleana utilizzata per indicare se un utente e' loggato o no
17. bool print -> variabile booleana utilizzata per indicare se stampare le informazioni dell'utente o no

Per salvare le informazioni dell'idle time utilizzo una struct in cui salvo, l'ora, i minuti e i secondi del tempo di attivita' del dispositivo di comunicazione utilizzata da un utente.

Per quanto riguarda la formattazione secondo l'opzione -s, utilizzo una struct chiama **indent_s** in cui salvo il nome utente, il nome reale e la stringa del tty piu' lunghe per impostare l'indentazione della "tabella".

Ricerca degli utenti/processi

Il comando, se non vengono specificati utenti, stampa tutti gli utenti loggati in base al comando specificato (se -s o -l in quanto sono le due opzioni che settano la formattazione dei dati); se non venogno specificati neanche le opzioni, allora, il comanda stampa le informazioni di tutti gli utenti loggati secondo la fomattazione di -s.
Inizialmente effettua una ricerca di tutti gli utenti loggati attraverso la libreria utmp.h salvando le informazioni utili, e aggiungo ogni utente a un array in cui sono contenute tutti gli utenti secondo la struct da me creata, ovvero users_inf. Ad ogni iterazione per prendere gli utenti loggati attraverso la libreria utmp.h, effettuo un controllo sulla tipologia dell'utente in modo da assicurarmi che ottengo solo utenti loggati e ottengo tutte le informazioni mancanti dalla libreria pwd.h attraverso la funzione **getpwnam(nomeUtente)** che restituisce le informazioni dell'utente passto in input sotto forma di struct passwd in cui sono contenute le informazioni dell'utente (come la GECOS); esempio in pseudocodice:

```
# Funzione in cui salvo nell'array in posizione i le informazioni dell'utente
void logged_user(){
    struct utmp* ut;
    setutent();
    i = 0;
    while((ut = getutent()) != NULL){
        if(ut->ut_type == USER_PROCESS){
            users_inf_array[i].user_name = ut->ut_user;
            users_inf_array[i].tty = ut->ut_line;
            users_inf_array[i].host = ut->ut_host;
            users_inf_array[i].login_time = ut->ut_time;
            get_pwd_inf(users_inf_array[i].user_name, i);
            i ++;
        }
    }
    endutent();
}

# Funzione che dato un utente prende le informazioni relative di quell'utente con la libreria pwd.h
void get_pwd_inf(user_name, contatore i){
    struct passwd* pw = getpwnam(user_name);
    if(pw != NULL){
        users_inf_array[i].usr_dir = pw->pw_dir
        users_inf_array[i].shell_dir = pw->shell
        gecos[4] = take_gecos(pw->gecos)
        users_inf_array[i].real_name = gecos[0]
        users_inf_array[i].room_number = gecos[1]
        users_inf_array[i].office = gecos[2]
        users_inf_array[i].home = gecos[3]
    }
}
```

Per ottenere le GECOS separate utilizzo una funzione di appoggio in cui utilizzo la funzione **strtok(stringa, separatore)** per separare le varie informazioni e salvarle in un array di stringhe di dimensione 4 in cui avro' il nome reale come prima posizione (posizione 0), il room number come seconda posizione, l'office phone come terza posizione e infine l'home phone come ultima posizione; nel caso in cui una informazione non esiste assegno a quella possizione il valore *NULL* (esempio: non esiste l'office phone, allora viene assegnata come terza posizione dell'array il valore NULL).
Inoltre, invece di salvare le informazioni con un semplice '=' utilizzo la funzione **strdup(stringa da duplicare)**, tale funzione duplica la stringa presa in input e la assegna ad una variabile ed alloca dinamicamente la memoria necessaria per contenere una copia della stringa fornita come argomento e quindi copia il contenuto della stringa nella nuova area di memoria.
Una volta che salvo tutti gli utenti loggati, in base al tipo di formattazione specificata (se -s o -l) stampa le informazioni degli utenti loggati, solo nel caso in cui non sono stati specificati degli utenti.
Nel caso in cui vengono specificati degli utenti, viene effettuata una ricerca prima sugli utenti loggati salvati precedentemente in un array, nel caso in cui non vengono trovati allora l'utente non e' loggati e viene cercato tale utente in pwd.h sia attraverso il nome utente e il nome reale (quest'ultimo controllo viene effettuato solo se non viene specificato il comando -m).

Controllo nome reale

Per la ricerca attraverso il nome reale avviene solo quando viene specificato un nome negli argomenti in input; quando viene effettuato il controllo per la verifica dell'esistenza di un utente con tale nome viene sia controlla l'user name che il real name, per il controllo del real name utilizzo una funzione d'appoggio in quanto devo verificare se tale nome specificato e' contenuto nel nome reale (esempio: esiste un utente il cui nomea reale e' Gianni Rossi, e viene chiamato il comando finger rossi, viene restituito l'utente che ha real nome Gianni Rossi). Per la realizzazione della funzione prendo in input il nome specificato negli argomenti in input e il nome reale di un dato utente, utilizzo la funizone strtok(nome reale) per dividere il nome reale e controllo per ogni divisione se una parte del nome reale e' uguale al nome specificato in input.
Esempio del codice:

```
# Funzione che se il name_inp e' contenuto nel real_name restituisce true, altrimenti false;
void check_real_name(name_inp, real_name):
    char* token = strtok(real_name, " ");
    while(token != NULL){
        if(name_inp == token) return true
        token = strtok(NULL, " ");
    }
    return False;
```

Se viene specificato l'opzione -m questo controllo non viene eseguito.

Idle time

Per ottenere l'idle time di un utente loggato prendo da /dev il dispositivo di comunicazione utilizzato dall'utente loggato; tale file viene aperto attraverso la funzione **stat** tale funzione restituisce varie informazioni del file, come il tempo dell'ultimo accesso (salvato nella variabile st_atime), il tmepo dell'ultima modifica (salvato

nella variabile st_mtime) e il tempo dell'ultimo cambiamento di stato del file (salvato nella variabile st_ctime). Una volta aperto il file con tale funzione prendo il cambiamento dell'ultimo stato del file, ovvero il st_ctime, che restituisce un valore che sottrago al tempo corrente per ottenere quanti secondi sono passati dall'ultimo cambiamento di stato, che e' il nostro idle time. Una volta che ottengo la quantita in secondi effettuo dei calcoli per ottenre da quanti minuti/ore e' attivo il dispositivo di comunicazione e vengono salvati nella variabile idle_time dell'utente rappresentato attraverso la struct users_inf.

Mail e file .plan .projects .pgpkey

Per quanto riguarda il recupero delle informazioni della mail, eseguo lo stesso ragionamento utilizzato per l'idle time, salvo in last_mail e last_mail_read in cui salvo rispettivamente st_mtime (il tempo dell'ultima modifica) e st_atime (il tempo dell'ultimo accesso). Una volta salvati tali valori utilizzo la funzione **strftime** per salvare in un buffer il valore di qeuste variabile sotto forma di una stringa composta nel modo seguente:
"giornoDellaSettimana giorno Mese Ora:Minuti Anno"
Nel caso in cui last_mail e' maggiore di last_mail_read, nel caso stampa sia il last_mail che il last_mail_read, altrimenti stampa solo il last_mail_read.

Per quanto riguarda i file .plan .projects e .pgpkey vado a prendere i seguenti file nella direcotry principale dell'utente, apro i seguenti file attraverso la funzione fopen e in modalita' lettura iterando su ogni riga del file e stampando riga per riga, appogiandomi a una funzione per poter richiamarlo per entrambi i file.
Nel caso in cui viene specificato il comando -p il contenuto dei file non deve essere stampato (se specificato anche il comando -l)