RELAZIONE INCAPACHE 1.0/1.1 ANNO 2012/2013

BENEDETTI PAOLA CRESTA ROBERTA

Nel Main ...

Abbiamo cominciato a sviluppare questo laboratorio dal main, che riguardava soltanto la versione 1.0, inizializzando i file mreq_f e mansw_f con la funzione fdopen, rispettivamente in scrittura e lettura. In seguito abbiamo assegnato a [char * mycp] il path corrente, accertandoci che la sua grandezza fosse correttamente allocata.
Per far in modo che da questo punto in poi la radice del filesystem visibile da un utente fosse diversa da quella che il kernel usa realmente, ed invece uguale al path corrente, utilizziamo la funzione chroot su mycp. Questa prende come unico parametro un const char * e ritorna un intero, 0 nello specifico in caso di successo.
Nel nostro caso la nuova radice e' tmp, l'ambiente su cui lavoriamo, perché' ci permette di eseguire questa funzione, utilizzando i privilegi di root.
Specifichiamo le opzioni per il socket, di tipo stream, utilizzante il protocollo TCP, cioè inizializziamo tutta la struttura hints.
Proprio come il Ping/Pong, recuperiamo le informazioni necessarie per l'apertura di un socket server con una getaddrinfo, apriamo un socket con la funzione socket() e il valore che questa ritorna lo passiamo come primo argomento per la funzione bind(), mentre il secondo e terzo argomento sono rispettivamente l'indirizzo del socket e la dimensione della struttura che lo contiene. Infine chiamiamo la funzione listen() per mettere in ascolto il socket per eventuali connessioni (operazioni corretta perché il socket e' di tipo SOCK_STREAM).
myUID e' un numero intero che passiamo come unico argomento alla funzione setuid() per modificare il nostro UID che attualmente(essendo noi un processo con privilegi) e' 0, in modo che il cambio della radice sia definitivo.

Nell' incApache_http.c ...

/*Send_response*/
Questa funzione ha il compito di costruire l'header del file richiesto e di inviare sia header che file e usa da 4 a 6 parametri a seconda della versione da implementare:
-int out_socket (descrittore del file aperto in scrittura),
-int http_response (200, 404, ..),
-int http10 (per differenziare i protocolli 1.0 e 1.1),
-int my_thrd_no (numero di thread da passare in seguito alla join_prev_thread),

```
struct stat {
    dev_t     st_dev;     /* ID of device containing file */
    ino_t     st_ino;     /* inode number */
    mode_t    st_mode;    /* protection */
    nlink_t   st_nlink;   /* number of hard links */
    uid_t     st_uid;     /* user ID of owner */
    gid_t     st_gid;     /* group ID of owner */
    dev_t     st_rdev;    /* device ID (if special file) */
    off_t     st_size;    /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for file system I/O */
    blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
    time_t    st_atime;   /* time of last access */
    time_t    st_mtime;   /* time of last modification */
    time_t    st_ctime;   /* time of last status change */
};
```

-char* filename(nome del file richiesto),
-struct stat * stat_p (un puntatore a una struttura stat che viene riempita con le informazioni relative ad un file).

Veniva inizialmente richiesto di usare la funzione gmtime() che, preso un argomento di tipo puntatore a un time_t, restituisce un puntatore ad una struttura tm che è stata riempita con le informazione dell'argomento passato (il nostro è time_t now_t). Noi l'abbiamo usata all'interno di un'altra funzione(strftime()) con 4 argomenti: il primo è l'array di char my_time_string in cui vogliamo salvare il nostro formato di tempo e data; il secondo indica il numero massimo di caratteri che possono essere inseriti nella nostra stringa; il terzo il formato specifico che deve avere(per esempio %a (abbreviazione del giorno), %d(giorno), %b(mese), %Y(anno per intero)); il quarto un puntatore ad una struttura tm contenente le informazioni necessarie per formare la stringa, quindi proprio la nostra gmtime().
Un controllo su http10 ci assicura che in http_header venga inserito il protocollo giusto. Si entra quindi in uno switch su http_response.
Nel caso 200 veniva richiesto di riempire le variabili: unsigned long my_fsize e time_t fmot_t. Abbiamo utilizzato due variabili della struttura stat_p passata come argomento, ovvero, rispettivamente, st_size che rappresenta la grandezza totale del file in byte e st_mtime corrispondente al tempo di ultima modifica.
Nel caso 404 invece ci veniva chiesto di riempire le variabili mime_type, my_fsize e fmod_t. Dichiariamo una struct stat tmp (non *) e ci chiediamo se il file esiste e se è possibile aprirlo con un certo permesso attraverso la funzione access. In questo caso ci chiediamo se ERROR_404 ("404_Not_Found.html") esiste e in lettura; in caso affermativo possiamo riempire la struct tmp con le info del file ERROR_404 attraverso la funzione stat() ed inserire le info richieste in my_fsize e fmod_t usando tmp come nel caso 200. Per il mime_type invece, allochiamo uno spazio di memoria pari alla lunghezza di HTML_mime ("text/html") ed usiamo la strcpy per inserire il valore di HTML_mime in mime_type.

Per il caso 501 abbiamo agitato allo stesso modo.
Finito lo switch inizia la concatenazione su http_header di tutte le info necessarie, tra cui my_time_string, che nella richiesta succesiva viene sovrascritta utilizzando la strftime, ma usando localtime() invece di gmtime().
Infine si richiede di eseguire la sendfile, che prende 4 argomenti e copia dati tra i 2 file descriptor :
-out_socket (fd aperto in scrittura)
-my_fd (fc aperto in lettura)
-NULL (se (off_t*) offset è NULL, i dati saranno letti da my_fd partendo dal corrente file offset(concetto legato alla gestione degli array e dell'aritmetica dei puntatori ed indica il numero di byte da aggiungere ad un indirizzo di base per ottenerne uno assoluto))
-my_fsize (numero di byte da copiare).

/*Manage_http_requests*/
Questa funzione ha il compito di riconoscere e valutare i metodi che vengono richiesti per così essere in grado di richiamare la funzione appena descritta send_response con i parametri corretti.
Viene chiamata a sua volta dalla funzione client_connection_thread con due parametri, un intero corrispondente al socket e, nel caso della versione 1.1, un intero corrispondente al numero di connessione.
Il primo compito è stato quello di interpretare la riga di richiesta inserita dall'utente, separandola in ter parti distinte: il metodo, il nome del file richiesto e il protocollo da usare.
Usiamo la funzione strtok_r su (char *) http_request_line, usando il delimitatore " " e un puntatore (char * strtokr_save)ad una variabile char * che viene utilizzato internamente da strtok_r () per mantenere il contesto tra le chiamate successive che analizzano la stessa stringa. Alla prima chiamata, http_request_line punterà alla stringa che cui bisogna eseguire il parser e il valore di strtokr_save viene ignorato; nelle chiamate successive il primo argomento di strtok_r sarà NULL, mentre il delimitatore sarà " " per filename, "\r\n" per protocol. Ora in my_method c'è il comando che il nostro utente ha inserito, su cui noi possiamo fare dei controlli. Nel ciclo for in cui stampiamo le option line, se il metodo non è conditional, cerchiamo la stringa "If-Modified-Since: ... " nella http_option_line e se abbiamo successo, creiamo un puntatore ad una stringa date in cui salviamo il token della http_option_line fino a che non troviamo il separatore "\r\n".
In seguito, con la strptime, convertiamo il puntatore alla stringa date in valore che sono salvati nella struttura since_tm, usando il formato espresso nel secondo parametro(come strftime). Poniamo my_method in or bit a bit con METHOD_CONDITIONAL per dargli la possibilità di entrare nell'else if, dopo che si sono già fatti i controlli su 501 e 404.
infine salviamo sella variabile time_t tmp1 il valore corrispondente alla data e al tempo della struttura since_tm che

passiamo per riferimento alla funzione mktime.
Nel caso my_method non sia né 501, né 404, e sia conditional, dopo aver controllato l'esistenza del file, dei permessi e dopo aver iniziazzato la struttura stat_p con le info del file, calcoliamo la differenza tra tmp1 e stat_p->st_mtime con la funzione difftime.
Se la differenza risulta 0, allora my_method diventerà NOTCHANGED cosicché, nello switch immediatamente dopo, la richiesta entri nel case 304.

Nell' inc Apache_thread.c ...

all'interno di questo file dovevamo organizzare le strutture array che ci permettessero di gestire le threads e il reeindirizzamento dei loro output in modo che il ostro server potesse gestire piu connessioni attive sulla stessa porta.
Per fare questo, all'interno della client_connection_thread abbiamo inizializzato dell'array di puntatore ad elementi thread_id:to_join in modo che il suo primo elemento sia un puntatore a null.
Le funzioni che ci sono state date da implementare in questo file, svolgono fondamentalmente un azione molto simile ma a due livelli diversi.
La jpoin_prev_thread deve attendere che le threads precedenti a quella identificata dall'intero passato come parametro alla funzione terminino.
Per fare questo ,prima di tutto controllo che non si sia nel caso in cui si stia cercando la thread precedente del primo thread lanciato. Dopodiché attendo che la thread precedente termini, incremento il numero di threads attualmente libere, risetto a -1 il contatore per le thread attive per una determinata connessione e decremento il contatore che tiene conto del numero di thread da processare. Tutto questo dopo aver bloccato il mutex in modo che non ci siano problemi di coerenza. La join all thread invece, teoricamente dovrebbe far si che tutte le thread di una determinata connessione terminino; in pratica poi deve attendere che la prima thread lanciata termini, il che significa per induzione che anche le altre sono terminate, libera la connessione e decrementa il numero di thread attive per il numero di connessione passato alla funzione come parametro che adesso avrà valore zero. In questo caso abbiamo un unica istruzione che va a modificare la parte di codice in comune fra i vari processi, il che ci permette di non fare utilizzo del mutex.

Infine, alla riga 266 e 267 dovevamo strutturare la coda to_join in modo che le per un determinato identificatore di thread, la cella corrispondente puntasse esattamente alla thread precedente a quella specificata.