Oggi è presentato il terzo laboratorio. Tuttavia, prima le basi teoriche per poterlo fare.

Vediamo innanzitutto la rappresentazione del tempo all’interno di un sistema di calcolo. Questo argomento apparentemente semplice comporta una serie di chiarimenti che devono essere fatti. Dal punto di vista hardware nel nostro sistema di calcolo tipicamente abbiamo un orologio vicino al processore. L’idea sarebbe quindi quella di effettuare dei cicli di lettura sul dispositivo in modo da ricavare l’ora esatta, ma la strada che è stata seguita è diversa. Di clock in un sistema di calcolo ce ne sono tanti, quello di cui parliamo adesso serve unicamente a tenere conto dell’avanzamento dell’ora: la sua differenza rispetto agli orologi classici è che non c’è un display.   
L’accesso all’orologio da parte del sistema è un’operazione lenta, perché storicamente è stato così. Questo da problemi molteplici: innanzitutto dal punto di vista dell’efficienza e poi dal fatto che nel momento in cui si acquisisce un’ora, il risultato non è più corretto (perché è passato del tempo). Quindi normalmente il tempo non viene rappresentato sulla base dell’orologio, ma viene utilizzato un orologio software: da qualche parte in memoria ci saranno una o più celle dedicate a conservare l’ora: la memoria RAM infatti seppur più lenta del processore è più veloce del clock per l’accesso al suo contenuto. Come si fa ad aggiornare continuamente il valore dell’orologio software al passare del tempo? Tramite dei segnali di interruzione che funzionano da sveglie per il processore (sono detti timer infatti). Quindi l’orologio hardware inizializza l’orologio software e dopodichè ogni tot. Volte al secondo un timer invia al processore un segnale per incrementare il contenuto dei registri dell’orologio software. Essendo software, è il sistema operativo a decidere quanto preciso dovrà essere questo orologio e, allo stesso tempo, quante interruzioni al secondo inviare al processore.

Lo scopo dell’orologio hardware serve quindi principalmente a tenere conto dello scorrere del tempo quando il dispositivo è spento. Tipicamente è alimentato tramite una piccola batteria.

La rappresentazione interna dei registri dell’orologio software è compatta, sotto forma di numero, il cui significato è contare la quantità di tempo passata da un istante di tempo passato ben definito, chiamato “Epoca” (EPOCH).   
Ci sono varie rappresentazioni dell’Epoca, ne vedremo 2. La prima è quella adottata dai sistemi UNIX: in questo standard il contatore presente in RAM è un po’ complicato e richiede l’uso di due celle di memoria RAM. Dal punto di vista dell’interfaccia di programmazione C viene definito con una struttura dati di tipo struct chiamata “timespec” che contiene due elementi: time-t tv\_sec; long tv\_nsec. Nel complesso questi due numeri rappresentano la quantità di tempo passata rispetto all’epoca. L’EPOCA per convenzione parte dalla mezzanotte (0:00) del 1 gennaio 1970. Tv\_sec rappresenta il numero di secondi passati rispetto all’epoca, tv\_nsec rappresenta le frazioni di secondo passate in più rispetto ai secondi interi, in nanosecondi (anche se il tempo non è mai preciso al nanosecondo per via del load iniziale dall’orologio fisico). Questo permette di distinguere un nanosecondo dall’altro senza arrivare a una rappresentazione binaria eccessiva. Time\_t è un tipo analogo agli int, la variabile time\_t è un numero intero con segno su 32 bit: il numero massimo rappresentabile è 2^31-1. La data che corrisponde all’Overflow di questo sistema di rappresentazioni è l’anno 2036. Questa era una delle rappresentazioni standard.

L’altra rappresentazione standard è la UTC (usata all’interno di alcuni protocolli di comunicazione, come L’NTP, un protocollo destinato a ottenere la sincronizzazione degli orologi interni sfruttando la comunicazione via rete). La rappresentazione UTC è simile a quella UNIX: anch’essa prevede l’utilizzo di due variabili da 32 bit, una per i secondi e l’altra per i nanosecondi. Ci sono però alcune differenze: Mentre il sistema UNIX fa partire l’EPOCA dal 1970, L’UTC la fa partire dal 1900 (sempre il primo gennaio a mezzanotte). Inoltre il sistema UTC rappresenta i secondi come interi senza segno (quindi si possono arrivare a contare fino a 2^32-1 secondi). L’intervallo degli anni rappresentabili non è molto diverso per via dello spostamento dell’epoca.

Il primo metodo è usato internamente dai sistemi di calcolo, il secondo dai protocolli di comunicazione tra dispositivi tramite rete.

Per passare a qualcosa che sia leggibile da una persona senza troppe difficoltà vengono introdotte altre rappresentazioni del tempo. Innanzitutto c’è la rappresentazione testuale, che fa riferimento ai caratteri ASCII: sono anche stati definiti degli standard, in particolare c’è lo standard ISO-8601 a cui corrisponde anche un altro standard molto simile (ma leggermente diverso), L’RFC-3339. L’idea alla base dello standard ISO era quello di cercare di unificare i modi di rappresentare le date a livello mondiale (ogni Paese ha sempre fatto a modo proprio, per esempio se per noi 08/04 è l’8 aprile, per gli americani questo è il 4 di agosto). Lo standard ISO prevede che la data sia rappresentata in più formati predefiniti, tra cui quello di base è del tipo yyyy mm dd (4 caratteri per l’anno seguiti da due per il mese e due per il giorno). La data di oggi secondo lo standard ISO sarebbe 20210408. Una rappresentazione alternativa leggermente meno compatta introduce i trattini tra i numeri. Per quanto riguarda l’ora si usando due caratteri per l’ora, due per i minuti, due per i secondi e fino a tre per le frazioni di secondo: HH MM SS NNN. Le ore sono rappresentate su base 24, e in una rappresentazione non compatta presenta i due punti : tra i valori di ore, minuti e secondi. Se si vuole rappresentare sia il giorno sia l’ora l’operatore per separare i giorni dalle ore è il separatore T (es. 2021-04-01 T 14:48:33).

Un problema è sollevato dal diverso fuso orario in cui ci si trova, Si potrebbe optare sia per una rappresentazione assoluta e uniforme dell’ora (l’UTC) sia per una versione locale comprendente il fuso orario. L’orario UTC è indicato con la lettera maiuscola Z alla fine della rappresentazione dell’ora, altrimenti va specificato il fuso orario (ad esempio con 02:00).

L’RFC-3339 invece fa parte degli standard utilizzati sulla rete internet, questo standard è ispirato all’ISO-8601, ma presenta delle differenze. La differenza principale è che il separatore T può essere sostituito da uno spazio vuoto, rendendo la lettura dell’ora più facile per un essere umano. A parte sottilissime differenze, quindi, la rappresentazione dell’ora secondo questo standard è quasi uguale a quella dell’ISO. Ci sono delle varianti sulla rappresentazioni dei secondi: anziché essere tra 0 e 59 i secondi vanno da 0 a 60, per poter introdurre quelle correzioni usate ogni tot. Anni per riaggiustare l’ora esatta rispetto al tempo di rotazione della Terra attorno al suo asse.

C’è una via di mezzo della quale si può tenere conto (tra la rappresentazione binaria e quella leggibile sotto forma di stringa ASCII strutturata secondo uno standard): tale via di mezzo è una rappresentazione interna ma strutturata in maniera simile agli standard di lettura. Questa rappresentazione media è ottenuta definendo una struct tm, attraverso una struct di valori che occupano separatamente l’anno, il mese, il giorno, l’ora, i minuti e i secondi. Infatti è strutturata in questo modo:

struct tm {

int tm\_sec;

int tm\_min;

int tm\_hour;

…

};

Questa modalità è detta “broken down”.

Il range dei valori è analoga alla rappresentazione testuale (0-60 per i secondi, 0-59 per i minuti, 0 -23 per le ore ecc.). La differenza è che anziché utilizzare dei caratteri ASCII vengono usati i campi della struct, il che rende più facile la comprensione da parte del sistema di calcolo. A parte queste strutture, sono definite delle librerie per la conversione delle date da un formato all’altro.

Il secondo argomento del laboratorio di oggi riguarda l’accesso ai file in ambiente POSIX. In particolare, vedremo l’accesso ai file in modalità scrittura. Un problema che si pone in questo caso è quello di potenziali “corse critiche” (race conditions).

Supponiamo di avere due applicazioni A1, A2 e che vogliano entrambe accedere allo stesso file F. Se le due applicazioni vogliono entrambe effettuare una lettura non c’è nessun problema a condividere il file, ma nel caso in cui invece le applicazioni vogliano effettuare dei cambiamenti sul file si possono disturbare a vicenda. Non c’è nel file system un sistema di sicurezza per evitare questo: per l’accesso al file entrambe le operazioni usano la call “open” e possono entrambe accedere con lo stesso diritto allo stesso file. Ma questo può causare delle condizioni di errore (anche gravi). Supponiamo che A1 voglia scrivere una certa cosa e A2 una cosa diversa, all’interno dello stesso file. Quello che potremmo aspettarci all’interno del file è che se A1 viene eseguita prima allora nel file ci troveremo il risultato di A2 e viceversa, ma non è sempre così: Può succedere che il file F venga corrotto dalla scrittura multipla e quindi non vi si trovi più il risultato di nessuna delle due applicazioni. Questo dipende da come viene implementata la scrittura all’interno dei file (che normalmente è Bufferizzata). Può capitare che le aree di memorie uste per bufferare l’output scrivano contemporaneamente sul file in memoria di massa, dunque alla fine l’output sarà un miscuglio Casuale del risultato delle due operazioni (quindi c’è anche la possibilità che vada tutto bene per puro caso). Per evitare le corse critiche è trovare il modo di sincronizzare le varie operazioni in modo che quando una sta effettuando la modifica di un file l’altra non possa interagire: questo è ottenuto attraverso un meccanismo di “locking”. Quando si accede a un file per effettuarne la scrittura la nostra Applicazione A1 deve fissare un lock sul file F e rilasciarlo solo DOPO aver terminato le operazioni su di esso. Detta così sembra semplice, in realtà se si guarda l’implementazione POSIX dei meccanismi di locking le cose si complicano. Le difficoltà sono in parte legate alla difficoltà di utilizzo dell’API e in parte dai bug presenti nei vari sistemi. Una soluzione che più o meno funzione è quello che vedremo oggi ed è quello degli advisory lock (secondo l’implementazione POSIX).

Advisory (contrapposto a mandatoy) indica che può essere usato dall’applicazione per evitare corse critiche a patto che Tutte le applicazioni decidano di adottare tale meccanismo (se un’applicazione sceglie di fare dei danni comunque può farli). Il meccanismo è descritto sul manuale (guarda nel file README del laboratorio).

L’applicazione proposta nel laboratorio serve ad inserire dei messaggi all’interno di un file di Log. Un file di Log è un file testuale gestito in modalità append, in cui varie applicazioni inseriscono delle registrazioni. L’utilità di questi file di log è molto alta perché permettono di riconoscere anomalie di funzionamento.  
Ogni registrazione viene etichettata con una data e un’ora (in modo da poter ricostruire cosa è sucesso in ordine cronologico).   
Ogni riga inizia con una data, seguita da una stringa di caratteri che identifica l’evento da inserire all’interno del file di log. Ovviamente vogliamo evitare corse critiche in modo da preservare l’ordine delle informazioni del file di log, la data serve perché i file di log possono tenere milioni di righe e quindi serve per permettere un parsing automatico delle informazioni in esso salvate.

Il programma contiene un file write\_log.c, che va completato per farlo funzionare. Le parti mancanti sono in parte riferite alla rappresentazione delle date e in parte al meccanismo di interazione in scrittura su uno stesso file da parte di più applicazioni (e fa riferimento ai lock di tipo Avisory).

All’avvio il programma deve essere chiamato seguito da un commento senza spazi, da essere inserito all’interno del file di log. La data corrente viene generata e inserita in automatico attraverso il valore interno dell’orologio, ottenuto attraverso una system call, che viene convertita nello standard RFC-3339 (passando dalla rappresentazione binaria della struct Timespec alla rappresentazione broken down della struct tm, dalla quale una funzione di libreria estrae i valori e crea la stringa in formato standard. Il formato di funzionamento di questa funzione di libreria è simile al printf (ci sono delle cose analoghe al %s,%d ecc. che servono per essere rimpiazzati dal numero dell’anno, del mese e del giorno).  
Un’altra piccola difficoltà è che viene usata una stringa di caratteri di dimensioni predefinite (viene definita una char string[const]) e la cosa da inserire deve essere contenuta all’interno di questo array di caratteri; se dovesse debordare, il commento deve essere troncato.

Ovviamente prima della modifica del file Log, quindi, la nostra applicazione dovrà acquisire il lock (chiamando la fcntl). Se c’è un’altra applicazione che ha già messo il lock l’esecuzione del programma che sta provando ad accedere al file viene ritardata.  
Questi sono programmi che richiedono un brevissimo tempo di esecuzione, quindi non ci srebbe mai la possibilità di verificare il corretto funzionamento del meccanismo di lock viene aggiunta la possibilità di ritardare la conclusione dell’applicazione, inserendo un terzo elemento sulla riga di commento (non importa cosa, basta che ci sia scritto qualcosa, perché il programma avvia la funzione ritardata appena nota che argc > 2). Il consenso dell’utente per terminare è dato con un a capo ricevuto dal terminale. Questo permette di mandare la write\_log in esecuzione su più shell diverse con un commento e un terzo parametro e poter così osservare che i processi avviati dopo restano in attesa finché non do il consenso di terminazione al primo programma, dopodiché parte il secondo, poi se c’è il terzo e così via.

Un altro aspetto che richiede attenzione è l’andare a verificare le condizioni di errore nelle chiamate di sistema: le system call segnalano errori ritornando il valore costante -1, che quindi deve essere memorizzato e controllato per verificare che non sia successo qualcosa di sbagliato (normalmente ritornano il valore 0). Il -1 indica solo che c’è stato un errore, se si vuole sapere quale errore si è verificato bisogna guardare in un’altra variabile di tipo intero, chiamato errno. Questo valore può essere stampato in output attraverso la funzione perr().

La funzione fcntl ritorna solo se è possibile ottenere il lock, sennò fa restare in attesa. Chiamando la open si ottiene il file descriptor che va inserito nella fcntl assieme al lock da inserire. Si può richiamare la stessa funzione per togliere il lock, ma non serve perché viene tolto in automatico quando si chiama la close.

C’è anche una parte opzionale per settimana prossima.