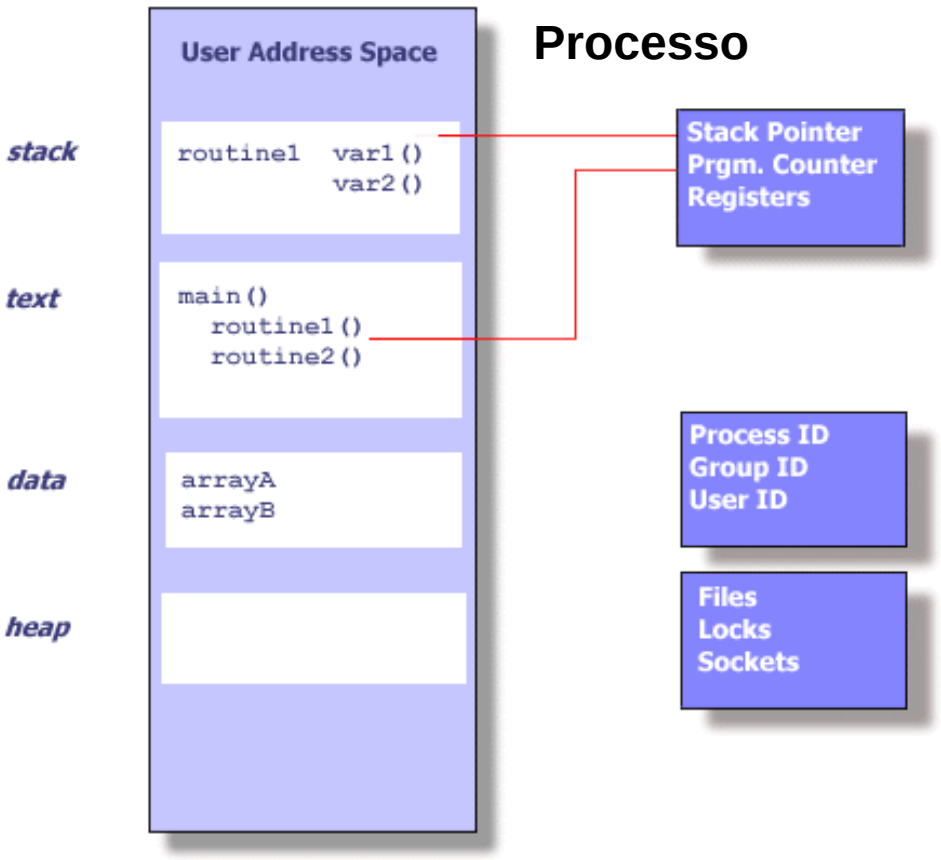
I THREAD

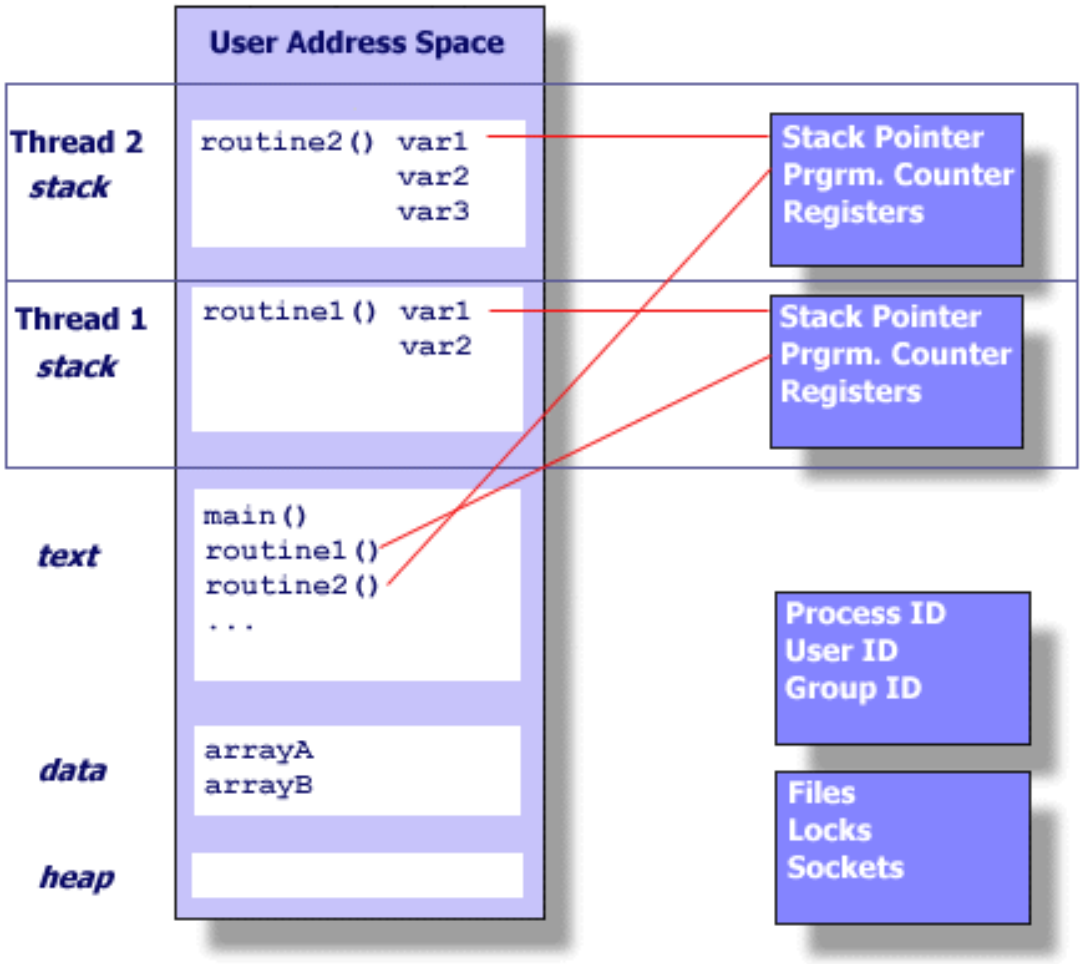
- **Un thread è un singolo flusso di istruzioni, all'interno di un processo, che lo scheduler può fare eseguire separatamente e concorrentemente con il resto del processo.**
- Un thread può essere pensato come una procedura che lavora in parallelo con altre procedure all'interno di un processo
- Per eseguire in parallelo con il resto del processo, un thread deve possedere delle strutture dati (un proprio contesto) cioè un proprio ambiente di esecuzione, per realizzare il proprio flusso di controllo.
- **Ogni processo ha il proprio contesto**, ovvero il proprio Process ID, Program Counter, Stato dei Registri, Stack, Codice, Dati, File Descriptors, Entità IPC, Azioni dei segnali.
- Il Codice del processo è pensato per eseguire procedure sequenzialmente.
- L'astrazione dei thread vuole consentire di eseguire procedure in parallelo (concorrentemente), ovviamente scrivendo tali procedure in modo opportuno.
- **Ciascuna procedura da eseguire in parallelo sarà un thread**.



Il contesto di esecuzione di ciascun Thread

- Un thread è un singolo flusso di istruzioni, all'interno di un processo, che lo scheduler può fare eseguire separatamente e concorrentemente con il resto del processo. Per fare questo uno thread deve possedere un proprio contesto di esecuzione.
- Un Processo può avere più thread.
- Tutti i thread di uno stesso processo condividono le risorse del processo (dati globali e CPU) ed eseguono nello stesso spazio utente.
- Ciascun Thread può usare tutte le **variabili globali** del processo, e condivide **la tabella dei descrittori di file** del processo.
- Ciascun thread in più potrà avere anche dei **propri dati locali**, e sicuramente avrà un proprio **Stack**, un proprio **Program Counter**, un proprio **Stato dei Registri** ed una propria variabile **errno**.
- Dati globali ed entità del Thread (Dati, Stack, Codice, Program Counter, Stato dei Registi) rappresentano lo **stato di esecuzione del singolo thread**.

Processo con 2 thread



Vantaggi e Svantaggi nell'uso di Thread

- **Vantaggi**:
- Visibilità dei dati globali: condivisione di oggetti semplificata.
- Più flussi di esecuzione.
- gestione semplice di eventi asincroni (I/O per esempio)
- Comunicazioni veloci. Tutti i thread di un processo condividono lo stesso spazio di indirizzamento, quindi le comunicazioni tra thread sono più semplici delle comunicazioni tra processi.
- Context switch veloce. Nel passaggio da un thread ad un altro di uno stesso processo viene mantenuto buona parte dell'ambiente.
- **Svantaggi**:
- Concorrenza invece di parallelismo:
  - occorre gestire la mutua esclusione per evitare che più thread utilizzino in maniera scoordinata i dati condivisi, modificandoli in momenti sbagliati.
- La concorrenza deve essere gestita
  - sia da **chi scrive il programma** che usa i thread.
  - sia dal **sistema operativo che implementa le funzioni di libreria e le system calls utilizzate "contemporaneamente" da più thread di uno stesso processo**.
  - Le funzioni delle librerie di sistema e le system calls devono essere **rientranti** (**thread safe call**).

Operazioni Atomiche: definizione

il termine operazione qui è volutamente generico

- **In generale,** un'operazione che usa dei dati condivisi è **atomica** se è indivisibile, ovvero **se nessun'altra operazione**, che usa quegli stessi dati condivisi, **può cominciare prima che la prima sia finita**, e quindi non può esserci interleaving tra le diverse operazioni. Il risultato di quella operazione è sempre lo stesso se parte dalle stesse condizioni iniziali.
- Quando parleremo di istruzioni macchina che accedono alla memoria e introdurremo la paginazione e il page fault, vedremo che la parte **verde** di questa affermazione dovrà essere modificata e meglio precisata, ottenendo l'affermazione **blu** qui sotto:
- **un'operazione, che usa dati condivisi, è atomica, rispetto alle operazioni che usano gli stessi dati condivisi, se viene eseguita in modo indivisibile**.
  - **Può però capitare che una operazione indivisibile debba essere interrotta per permettere l'esecuzione di una operazione più urgente.**
  - **In caso di interruzione, un modo per rendere comunque l'operazione atomica consiste nell'attendere la fine dell'interruzione e far ripartire dall'inizio l'operazione interrotta che si voleva fosse eseguita atomicamente.**
  - **Poiché in caso di interruzione riparto da capo, le condizioni iniziali da cui l'operazione riparte potrebbero essere nel frattempo cambiate.**
  - **Ma, a parità di condizioni di partenza, il risultato di una operazione atomica sarà sempre lo stesso.**

Le istruzioni in linguaggio C sono Atomiche ?

- **Le istruzioni in linguaggio C sono atomiche? NO.**
  - in generale le istruzioni in linguaggio di alto livello non sono atomiche.
- **Le istruzioni di solo assegnamento di una costante ad una variabile e le istruzioni di sola lettura di una variabile, sono atomiche? NO**
  - **Sono atomiche solo se la dimensione della variabile è minore dell'ampiezza del bus dati.**
  - Ad esempio, in un processore i386, con bus dati a 32 bit, l'assegnamento seguente viene addirittura tradotto in DUE istruzioni assembly.

    (vedere test_assegnamento_var_grande.c)

```
uint64_t G;
int main()
{
        G=2034573288479;

}
```

```
mov     DWORD PTR G,   -1241209825
mov     DWORD PTR G+4, 473
```

NB: Se non disponete di un processore con bus dati avente ampiezza a 32 bit, potete comunque generare il codice così se compilate con il flag -m32 ed aggiungete il flag -S per generare l'assembly.