Mappatura di pagine in sistemi UNIX

Noi abbiamo un address space, in cui in mezzo non ci sono pagine mappate. Quindi come già detto, se noi esprimiamo un indirizzo che casca li dentro abbiamo un segmentation fault.
Come possiamo mappare queste pagine?
Il servizio che abbiamo a disposizione per fare ciò si chiama MMAP (Memory Mapping).

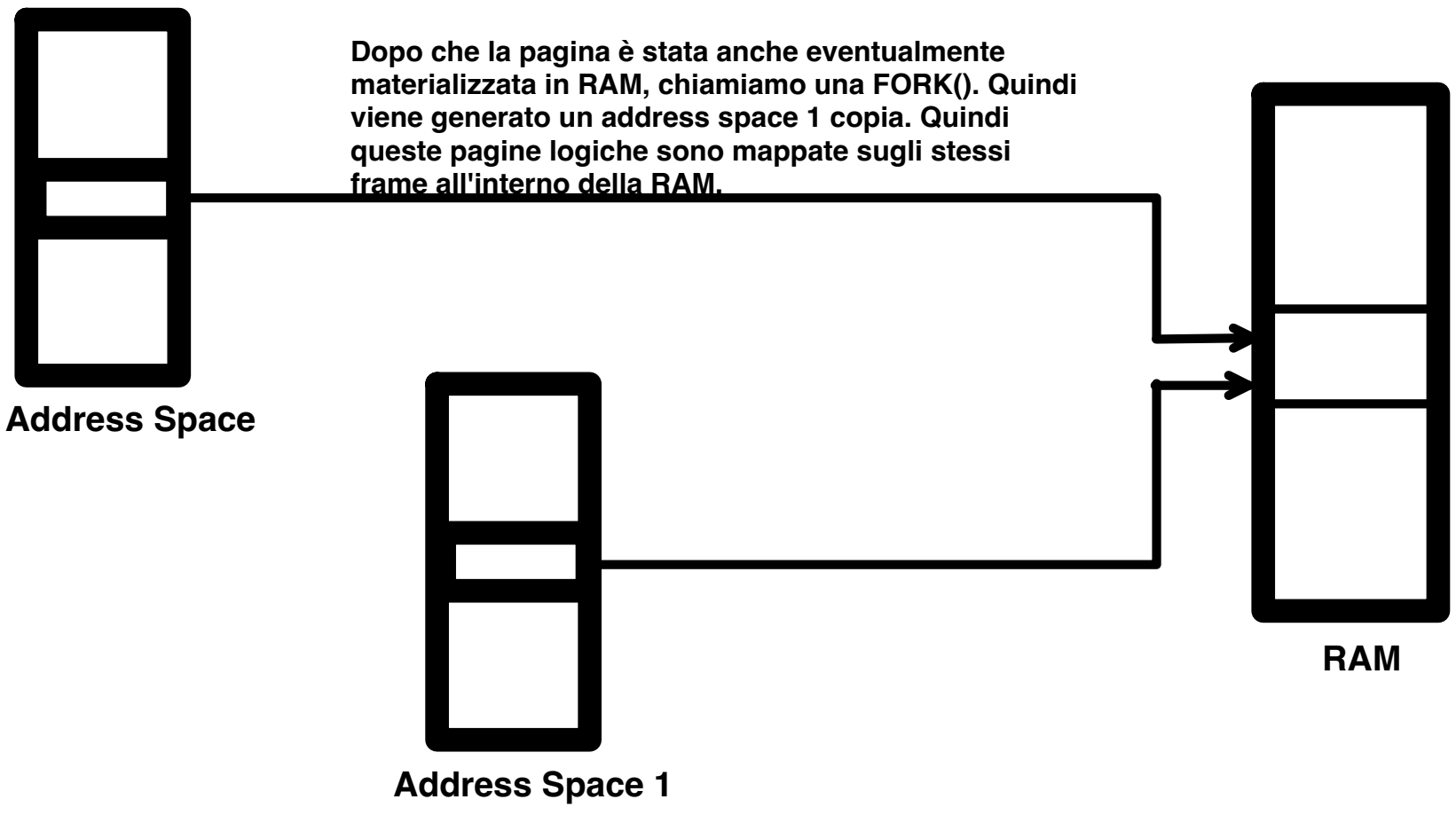
```
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset)
```

Descrizione valida (mappa) una regione di pagine contigue

Argomenti
1) *addr: inizio della regione da mappare (con NULL sceglie il kernel)
2) lenght: taglia della regione da mappare
3) prot: tipo di accesso desiderato (PROT_EXEC, PROT_READ, PROT_WRITE, PROT_NONE)
4) flags: opzioni (MAP_PRIVATE, MAP_SHARED, MAP_ANONYMOUS ..)
5) fd: descrittore verso un file da mappare nelle pagine
6) offset: posizione del file pointer da dove iniziare la mappatura del file

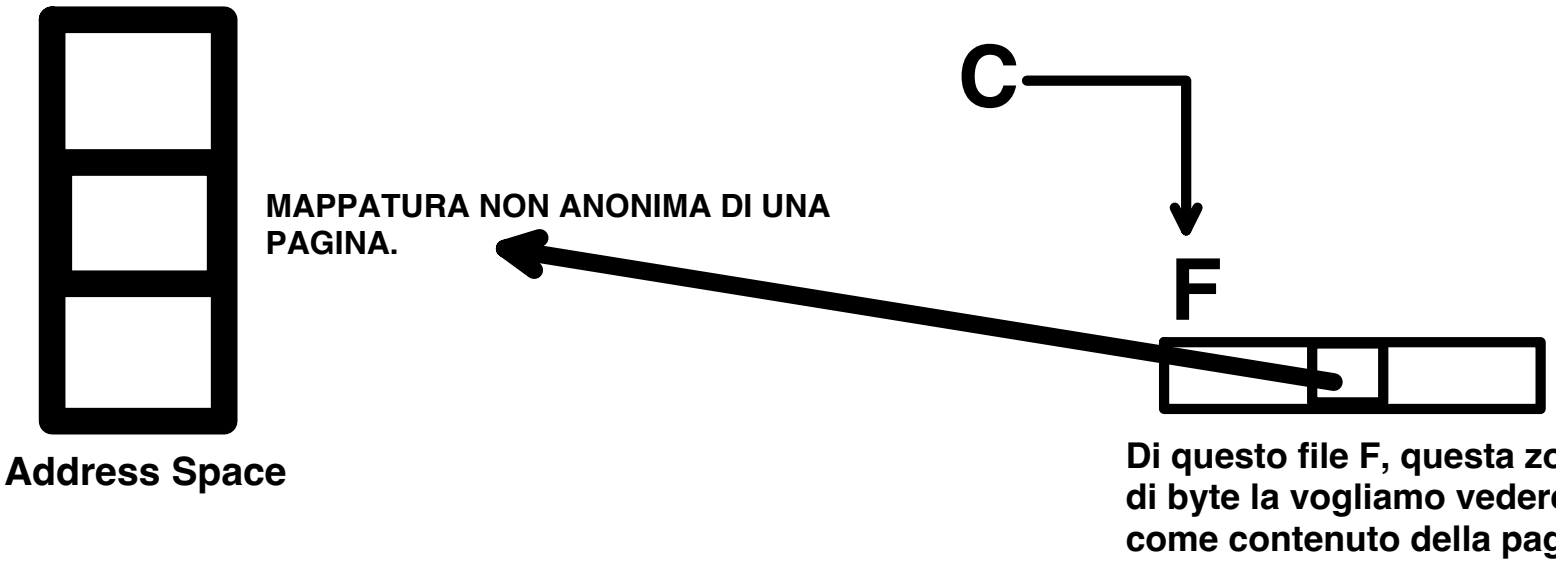
Restituzione NULL in caso di fallimento

Questo servizio restituirà un indirizzo che è l'indirizzo della ZONA che è stata mappata.
Supponiamo di avere un address space, nella zona delle pagine non mappate, noi andiamo a chiedere l'utilizzo di alcune pagine, magari 2 pagine all'interno di quella zona.
Ci verrà restituito l'indirizzo (la posizione) della prima di queste due pagine mappate, in particolare ci verrà restituito l'indirizzo logico I che noi possiamo utilizzare per accedere all'interno di questo contenitore. Come quando chiamiamo una malloc.
Il secondo parametro specifica quant'è la memoria che vogliamo mappare, quindi la taglia della regione da mappare, ma questo valore sarà arrotondato alla taglia della pagina tale per cui se noi chiediamo la size di una pagina e mezza (abbiamo detto che nei sistemi moderni le pagine sono di 4K, ma noi stiamo chiedendo di mappare 6K) dato che l'address space viene gestito a livello di pagine, ci viene restituito l'indirizzo di una coppia di pagine che possiamo utilizzare.
Poi come terzo parametro possiamo specificare la protezione che vogliamo applicare (Protezione lettura, scrittura, esecuzione, none-> voglio che le pagine siano mappate ma inutilizzabili).
Avremo anche dei flags che ci indicano la regola per gestire queste pagine, possiamo avere l'indicazione di pagine private o pagine condivise, oppure anonime.
Supponiamo di andare a richiedere l'utilizzo di una pagina logica all'interno dell'address space e il sistema ci restituisce qual è la posizione dove noi possiamo lavorare, quindi dove è presente quella pagina all'interno dell'address space.
Supponiamo che questa pagina sia SHARED:



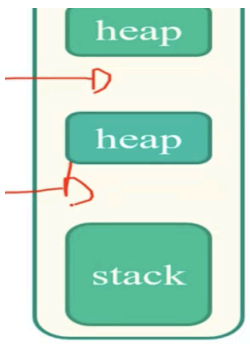
Questa pagina qui, che era stata mappata come shared, è una pagina "COPY ON WRITE" oppure no? Chiaramente NO. Quindi se il processo scrive all'interno della sua pagina logica e quindi aggiorna la memoria, queste informazioni possono essere viste/lette leggendo nell'address space 1 su questa pagina.
Noi stiamo realmente condividendo il contenuto della pagina fisica in RAM mappandoci due pagine logiche di due address space diversi in maniera shared.
Ovviamente se la pagina è privata essa è "COPY ON WRITE", ossia se qualcuno scrive il S.O intercetta questa scrittura e viene ad essere eseguita una copia privata e un aggiornamento delle page table su cui lavorare.
Quando una pagina è anonima, l'informazione che noi troveremo all'interno di queste pagine è null. Altrimenti cosa potremmo trovare quando cercheremo eventualmente di mappare una certa quantità di informazione, ma magari in modo "Non anonimo"?

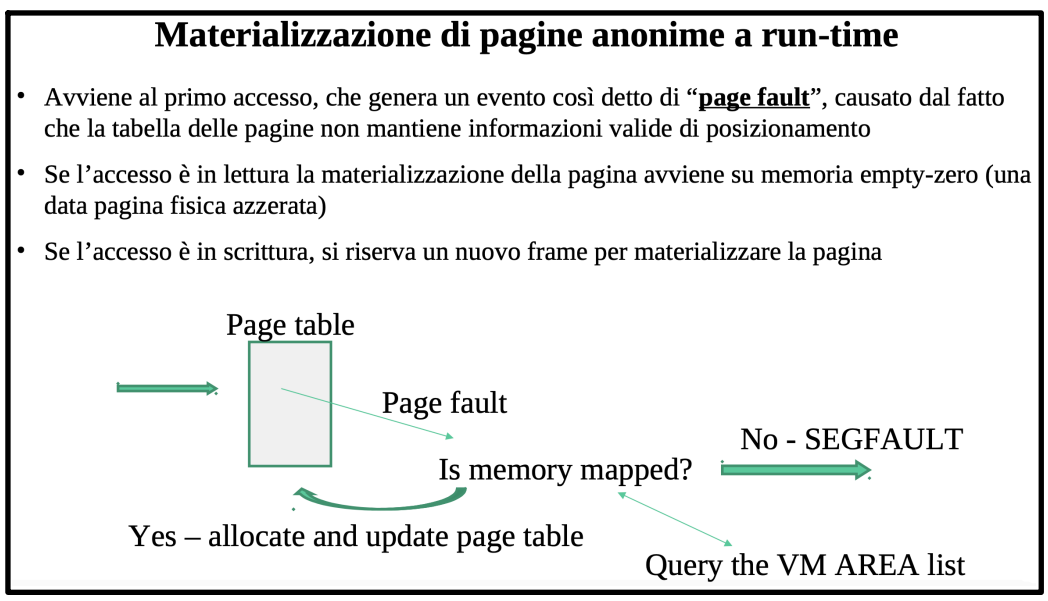
Con gli ultimi due parametri potremmo specificare, in particolare con un file descriptor che è un canale di I/O e con un offset, che in quella zona noi vogliamo vedere IL CONTENUTO DI UN FILE, in particolare a partire da un certo offset di quel file.
Il che implica dire che noi, con questa MMAP possiamo (all'interno di un AB) fare questo:



Quindi noi possiamo "memory mappare" contenuti che non sono anonimi, ma possono essere anche byte di file. I file descriptor non ci portano solo sui file, lo sappiamo, essi sono canali di I/O che ci portano in generale sugli oggetti di I/O che noi possiamo gestire, in questo caso la gestione la possiamo fare figurando che il contenuto di questi oggetti di I/O sia presente all'interno di alcune specifiche pagine che abbiamo all'interno del nostro address space.

Se noi cerchiamo di mappare pagina all'interno dell'address space IN UNA ZONA dove pagine mappate già ci sono, otteniamo un errore. Anche se il mapping è coperto da pagine già mappate e l'altra no. Quindi a cavallo in una zona tra mappate e non.
Quando noi abbiamo pagine anonime, ossia pagine il cui contenuto è essenzialmente NULLO, quando chiediamo di mapparle sono anche effettivamente materializzate in memoria fisica oppure no?
Ha poco senso che siano materializzate in memoria fisica, verranno materializzate solo quando - oltre a mapparle - noi le accederemo anche.
Quando noi andiamo a chiamare una MMAP per mappare pagine che sono anonime, queste pagine il cui contenuto è NULL, da NON MAPPATE ORIGINARIAMENTE diventano MAPPATE ma non materializzate, non ha senso materializzarle fino a quando l'applicazione non usa realmente la pagina logica e quindi finchè la CPU non deve accedere alla corrispettiva pagina fisica, questa cosa viene gestita dinamicamente in modo tale che le pagine anonime sono effettivamente materializzate all'interno della memoria di lavoro soltanto quando accade un cosiddetto PAGE FAULT, un errore sull'accesso alla memoria.
Noi stiamo esprimendo l'indirizzo logico per accedere alla memoria tale per cui all'interno della PAGE TABLE non c'è l'informazione di dove quell'indirizzo logico è presente nell'AB, quindi c'è un PAGE FAULT, e quindi questa eccezione riporta il controllo al software del kernel del sistema operativo;
Poi si chiede: Ma la pagina a cui l'applicazione sta accedendo, è mappata dentro l'address space, ma non ancora materializzata in RAM??
Se la risposta è SI, si sceglie un FRAME libero e si aggiorna la tabella delle pagine per portare la pagina logica verso quel frame.
Se la risposta è NO, abbiamo un segmentation fault. Stiamo cercando di accedere qui ad un indirizzo logico nell'AB che non è associato a nessuna pagina mappata all'interno dell'AB.



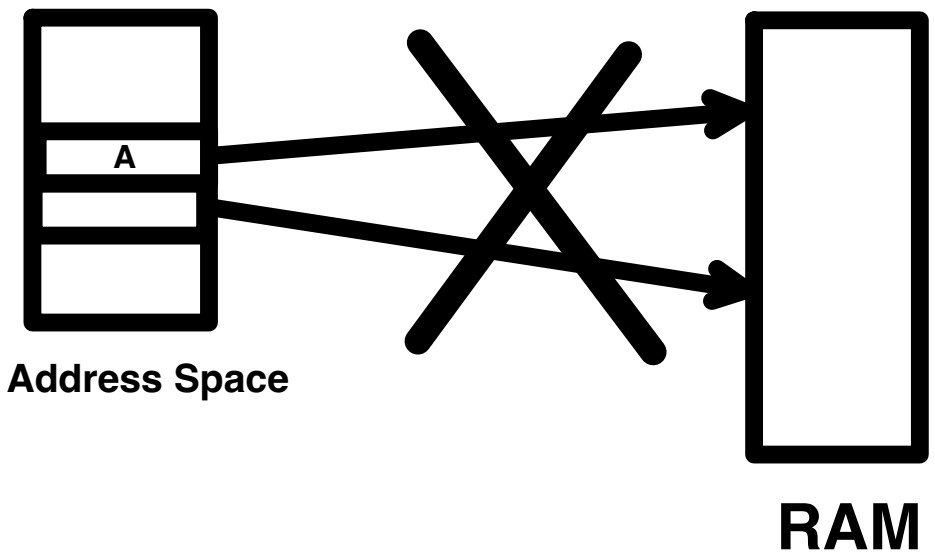


Se le pagine sono mappate nell'address space ma non materializzate in RAM ed io voglio accedervi con un offset dentro l'AB, non c'è problema, posso farlo, il nostro sistema operativo sceglie di andare ad allocarle in memoria fisica da qualche parte.

Ripetiamo:
-Se mappiamo la memoria come privata e poi eseguiamo una fork(), quelle pagine che sono state appena mappate sono protette da copy_on_write.
-Se mappiamo la memoria come shared, la rappresentazione delle pagine in RAM - e quindi la reale informazione che noi stiamo utilizzando - possono essere condivise con processi figli, quindi non è vero che la pagina è protetta copy_on_write, quindi uno scrive e l'altro può leggere i dati aggiornati, utilizzando address space differenti mappati sulla stessa pagina della memoria fisica.
-Se mappiamo come "non anonima" essa va ad indicare che vorremo che queste pagine, in realtà, nel momento in cui le andiamo ad utilizzare non abbiamo un contenuto semplicemente nullo, ma magari contengano qualche informazione e, tipicamente, con gli ultimi due parametri possiamo andare a spiegare che vogliamo che quelle pagine contengano un certo file di cui abbiamo un file descriptor (quindi questo significa che lo abbiamo aperto in precedenza quel file) e di quel file, vogliamo far comparire in quelle pagine esattamente una certa zona, di cui specifichiamo l'offset come parametro.
Attenzione: Quando noi mappiamo pagine con mmap il primo parametro - che è un indirizzo generico - può andare a specificare in maniera esplicita, all'interno dell'address space, in che punto queste pagine devono essere mappate.
Quindi stiamo scegliendo noi la posizione all'interno dell'address space.

Supponiamo che a me serva una memoria di 128 byte, e voglio ottenere l'utilizzo di questa memoria nel mio contenitore utilizzando una mmap.
La mmap lavora con la "granularità" pagina, il che implica dire che se io passo 128 come secondo parametro, quello che otterrò sarà perlomeno una pagina, perché ho chiesto una quantità di memoria da mappare che è contenibile all'interno di una pagina di 4KB.
Esiste quindi una sorta di discrepanza tra le operazioni che facciamo quando andiamo ad mmappare memoria passando una certa quantità di byte rispetto a quello che succede realmente in un address space: *quando passiamo una length in realtà vengono selezionate un numero di pagine sufficienti a contenere quella length*.

-Se noi passiamo una mappatura anonima, quindi stiamo dicendo che le pagine che sono all'interno dell'address space non devono contenere alcuna informazione - quindi dobbiamo semplicemente mantenerle per poterle utilizzare in futuro ma a contenuto inizialmente nullo - quando chiamiamo la mmap non è vero che quelle pagine vengono realmente anche materializzate in memoria.
Quindi quando nella mmap passiamo la mappatura anonima, dato un address space che avrà una zona di due pagine utilizzabile da ora in poi, NON è vero che queste pagine sono materializzate da qualche parte in RAM.



Questa cosa accadrà quando noi cominceremo ad utilizzare queste pagine, quindi quando cominceremo a voler leggere da A (all'inizio troveremo tutti NULL) oppure scrivere informazioni: chiaramente sappiamo che scrivere informazioni implica dire che la CPU esegue delle istruzioni macchina che devono accedere effettivamente alla memoria.
Ovviamente il processore esegue l'istruzione macchina in cui l'indirizzo che l'istruzione macchina vuole toccare, è un indirizzo all'interno della pagina, però poi sappiamo che andremo a toccare una specifica zona della RAM.

Però se questa pagina che stiamo cercando di toccare non è ancora stata materializzata in RAM, questa materializzazione avverrà non appena noi "tocchiamo" questa pagina.
Quindi la materializzazione effettiva delle pagine anonime avviene a RUN-TIME quando noi le andiamo realmente ad utilizzare.
Se noi cerchiamo di utilizzare un indirizzo logico associato ad una certa pagina P e nella nostra page table non c'è un'informazione nella entry associata a quella pagina che ci porta realmente su un frame, c'è un page fault (ossia una eccezione), il controllo va al sistema operativo che si chiede se a quella memoria a cui stavamo cercando di accedere fosse mappata o meno, se è mappata quella memoria deve essere utilizzabile e perciò ORA deve essere resa utilizzabile dal sistema operativo per farla accedere al software, e quindi verrà riservato un frame, verrà la page table per indicare qual è il frame su cui vogliamo andare, e siamo apposto. Poi il controllo ritorna all'applicazione che riesegue l'istruzione che ha generato questo fault, e accederà alla memoria.
Ovviamente se la pagina non è mappata abbiamo un segmentation fault, l'applicazione sta cercando di utilizzare la pagina logica che, attualmente -per quel contenitore- non è utilizzabile;
Ovviamente i segmentation fault possono possono avvenire quando noi cerchiamo di utilizzare questa pagina in modalità scrittura e magari questa pagina è protetta soltanto in modalità lettura o execute, che implica dire che nella page table c'è scritto qual è il frame associato alla pagina logica però, su quel frame, ci possiamo andare utilizzando soltanto operazioni di lettura o di fetch (istruzioni). Non possiamo andare a scrivere all'interno di quel frame.
Chiaramente questo prevede che all'interno della page table oltre ad avere l'informazione su dove la pagina logica è mappata in memoria fisica, abbiamo anche dei bit di protezione.