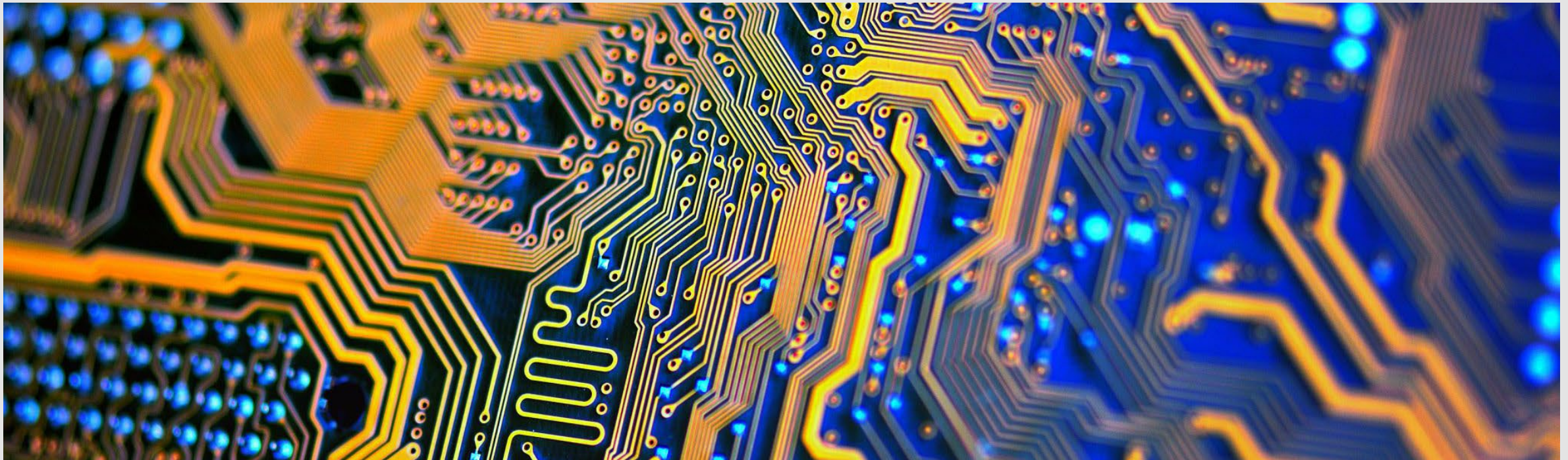


PROGRAMMAZIONE DI SISTEMA
A.A. 2022/2023

S318904 MARCELLO VITAGGIO
S317264 FABRIZIO VITALE

PINTOS – GESTIONE DELLA MEMORIA



PINTOS – SPAZIO DI INDIRIZZAMENTO

Fisico	Contenuto	Logico
00000000--000003ff	CPU	c0000000--800003ff
00000400--000005ff	BIOS	c0000400--800005ff
00000600--00007bff	Align	c0000600--80007bff
00007c00--00007dff	Pintos	c0007c00--80007dff
0000e000--0000efff	Pintos	c000e000--8000efff
0000f000--0000ffff	Pintos	c000f000--8000ffff
00010000--00020000	Pintos	c0010000--80020000
00020000--0009ffff	Pintos	c0020000--8009ffff
000a0000--000bffff	Video	c00a0000--800bffff
000c0000--000effff	Hardware	c00c0000--800effff
000f0000--000fffff	BIOS	c00f0000--800fffff
00100000--03ff ffff	Pintos	c0100000--c3ff ffff

- Lo spazio di indirizzamento è ben delineato tra spazio user e spazio kernel da `PHYS_BASE = 0xc0000000`
- Sopra `PHYS_BASE` l'address space (1GB) appartiene al kernel ed è mappato 1 a 1 con la memoria fisica, in altre parole, il kernel accede al primo indirizzo fisico 0x0 all'indirizzo logico `PHYS_BASE`.
- Lo spazio al di sotto di `PHYS_BASE` (3GB) è completamente mappabile dai processi utente

PINTOS - ALLOCAZIONE DI MEMORIA

- PintOS offre due metodi per l'allocazione di memoria, uno lavora con quantità di memoria rispetto a unità di pagine di memoria, mentre l'altro può allocare quantità di memoria di dimensione arbitraria, rispettivamente dichiarati come PAGE ALLOCATOR e BLOCK ALLOCATOR.

PINTOS - PAGE ALLOCATOR

- File: threads/palloc.c
- Viene utilizzato per allocare la memoria una pagina alla volta, ma può anche allocare più pagine contigue contemporaneamente.
- L'allocatore divide la memoria allocata (da 0x100000 fino a RAM_SIZE) in due pool, chiamati pool kernel e pool utente.
- Nella versione base di PINTOS, ogni allocazione viene effettuata dal pool del kernel.
- Per il tracking dello stato pagine, vengono utilizzate due bitmap, una per ciascun pool.

palloc_init():

```
uint8_t *free_start = ptov (1024 * 1024);
uint8_t *free_end = ptov (init_ram_pages * PGSIZE);
size_t free_pages = (free_end - free_start) / PGSIZE;
size_t user_pages = free_pages / 2;
size_t kernel_pages;
if (user_pages > user_page_limit)
    user_pages = user_page_limit;
kernel_pages = free_pages - user_pages;

/* Give half of memory to kernel, half to user. */
init_pool (&kernel_pool, free_start, kernel_pages, "kernel pool");
init_pool (&user_pool, free_start + kernel_pages * PGSIZE,
          user_pages, "user pool");
```

PINTOS – PAGE ALLOCATOR – METODI COINVOLTI

- `void *palloc_get_page (enum palloc_flags flags)`
- `void *palloc_get_multiple (enum palloc_flags flags, size_t page_cnt)`

Otengono e restituiscono rispettivamente una pagina o un numero di pagine contigue. Restituisce un puntatore nullo se le pagine non possono essere allocate.

L'argomento flag può essere una qualsiasi combinazione dei seguenti flag:
PAL_ASSERT, PAL_ZERO, PAL_USER

Rispettivamente utilizzati per:

- Generare un panic del kernel nel caso non sia stato possibile allocare le pagine volute (utilizzato solo durante la fase di inizializzazione del kernel).
- Azzerare il contenuto delle pagine prima di restituire le pagine
- Ottenere le pagine dallo user pool.

PINTOS – PAGE ALLOCATOR – METODI COINVOLTI

- `void palloc_free_page (void *page)`
- `void palloc_free_multiple (void *pages, size_t page_cnt)`

Liberano una pagina o un numero pari a *page_cnt* di pagine consecutive a partire da **pages*.

Vengono automaticamente chiamati al termine di un processo (stato exiting/dying).

Possono essere chiamati dallo stato di interruzione di un processo.

Vincolo: la zona di memoria da liberare deve essere stata originariamente ottenuta utilizzando uno dei due metodi del page allocator precedentemente descritti.

PINTOS – PAGE ALLOCATOR – METODI COINVOLTI

- File: bitmap.c
- `size_t bitmap_scan (const struct bitmap *, size_t start, size_t cnt, bool);`
- Policy: first fit

```
bitmap_scan (const struct bitmap *b, size_t start, size_t cnt, bool value)
{
    ASSERT (b != NULL);
    ASSERT (start <= b->bit_cnt);

    if (cnt <= b->bit_cnt)
    {
        size_t last = b->bit_cnt - cnt;
        size_t i;
        for (i = start; i <= last; i++)
            if (!bitmap_contains (b, i, cnt, !value))
                return i;
    }
    return BITMAP_ERROR;
}
```


PINTOS - BLOCK ALLOCATOR

- File: threads/malloc.c
- Si sovrappone all'allocatore di pagine descritto nella sezione precedente.
- Può essere utilizzato solo per allocazioni del kernel
- Prevede due diverse strategie di allocazione a seconda della dimensione del blocco ($< 0 >$ di 1 KB)
- La sintassi di quest

PINTOS - BLOCK ALLOCATOR - STRUTTURE

- Arena: rappresenta una zona di memoria contigua in cui vengono allocati i blocchi di una dimensione data dal corrispondente descriptor.
- Descriptor: mantiene le informazioni essenziali sulla natura dei blocchi e sull'arena
- Il block allocator define 2 particolari strutture per gestire i blocchi di varia dimensione:

```
struct arena
{
    unsigned magic;
    struct desc *desc;
    size_t free_cnt;
};
```

```
struct desc
{
    size_t block_size;
    size_t blocks_per_arena;
    struct list free_list;
    struct lock lock;
};
```

PINTOS - BLOCK ALLOCATOR - STRATEGIE

- Strategia 1 ($< 1\text{ kB}$): Le allocazioni vengono arrotondate alla potenza di 2 più vicina o a 16 byte, a seconda di quale sia il valore più grande. Quindi vengono raggruppate in una pagina utilizzata solo per allocazioni di quella dimensione.
- Strategia 2 ($< 1\text{ kB}$): Le allocazioni (più una piccola quantità di overhead) vengono arrotondate per eccesso a un multiplo della dimensione di pagina più vicino (minimo 4kB), l'allocatore di blocchi richiede quel numero di pagine contigue all'allocatore di pagine descritto precedentemente.

PINTOS - BLOCK ALLOCATOR – METODI COINVOLTI

- `void *malloc(size_t size)`
- `void *calloc(size_t a, size_t b)`

1. Trovare il descrittore più piccolo che soddisfa la richiesta di dimensione.
2. Acquisire il blocco associato al descrittore.
3. Se l'elenco libero è vuoto, creare una nuova arena, inizializzarla e aggiungere i suoi blocchi all'elenco libero.
4. Restituzione di un blocco dall'elenco libero.

I due metodi sono analoghi, con l'unica differenza che `calloc()` ricerca la dimensione minima pari a $a*b$ e utilizza `memset()` per inizializzare il contenuto del blocco a zero prima di restituirlo.

PINTOS - BLOCK ALLOCATOR – METODI COINVOLTI

- `void *realloc(void *old_block, size_t new_size)`

La funzione `realloc` tenta di ridimensionare un blocco a byte `new_size`, eventualmente spostandolo nel processo.

- In caso di successo, crea e restituisce un nuovo blocco, liberando quello vecchio.
- in caso di fallimento, restituisce un puntatore nullo. Il vecchio blocco rimane valido se la riallocazione fallisce.

PINTOS - BLOCK ALLOCATOR – METODI COINVOLTI

- `void free(void *p)`

Si occupa di liberare le porzioni di memoria precedentemente allocate dai metodi *malloc()* *calloc()* o *realloc()*, i blocchi vengono gestiti in maniera differente a seconda della loro dimensione:

- Per i blocchi normali, (*d->lock*) viene acquisito per garantire la corretta sincronizzazione con altri thread che accedono allo stesso descrittore. Il blocco viene aggiunto alla lista libera associata al descrittore. Se l'intera arena è inutilizzata (tutti i blocchi sono liberi), l'arena viene liberata.
- Per i blocchi grandi, dal momento che essi corrispondono esattamente a multipli di pagina (compreso overhead) viene direttamente chiamata la funzione del PAGE ALLOCATOR *palloc_free_multiple()*

COMPARAZIONE MEMORIA: PINTOS VS OS/161

PINTOS

- 64 MB RAM, non modificabile.
- Memoria vulnerabile a frammentazione interna (BA) e esterna (PA), le allocazioni di multipli di pagina devono essere limitate, mentre le allocazioni di pagina singola sono sempre terminate con successo, a meno di memoria completamente piena
- 80x86 non prevede una metodologia di accesso diretto tramite indirizzo fisico, limitazione aggirata dalla mappatura 1 a 1 spazio d'indirizzo logico fisico

OS/161

- Supporto previsto ma non implementato per i metodi di deallocazione della memoria e delle strutture necessarie al tracking dello stato della memoria (DUMBVM), l'esecuzione continua di processi continua ad allocare memoria senza mai rilasciarla.
- Quantità di RAM utilizzabile configurabile in sys161.conf
- Lo spazio di indirizzamento viene gestito in maniera molto simile a PINTOS.