Operating System Lab
Assignment 5

Multi-Level Scheduler

A Simple Round Robin Scheduler is implemented in Pintos. Each process runs for a fixed interval of time known as TIME SLICE or TIME QUANTA in a circular fashion.

1. When Pintos starts an idle thread is created which is to ensure that there is always at least one thread to schedule.
2. Thread is created using the function thread_create(), the thread is initialised with the thread's initial state initialised to THREAD_BLOCKED after which thread_create calls thread_unblock which unblocks the thread that is sets the state to THREAD_READY and puts it in the ready queue to be scheduled by the scheduler.
3. Timer interrupts occur TIMER_FREQ times per second. A call to thread_tick() is made which increases the running time of current thread . If the time quanta or slice of a running thread expires, a call to intr_yield_on_return() is made which ultimately results in a call to thread_yield().
4. thread_yield() picks the new thread to run using the function next_thread_to_run() and the actual context switch is done by the scheduler using the function thread_switch() which the context switches the current and the next thread.

5. thread_switch() stores the cpu's stack registers and the stack pointer in the struct of the current thread and loads the registers and the stack pointer of the next thread in the cpu's stack and cpu's stack pointer therefore restoring the new thread and returns and then thread_schedule_tail() is called.
6. thread_schedule_tail() changes the state to running. The previous thread's resources are released into the system if it was dying, it involves freeing of the page that contained the stack and the data structures of the dying thread.
7. Thus the context switch process finishes and the new thread starts running.

Data Structures that are added/modified

Added to the struct thread in thread.h -

1. Int qno - This denotes the current queue the thread is in, (0, 1) → (L1, L2)
2. Int total_time - This field has different meaning depending on the current queue the thread is in. If the thread is in L1 then total time denotes the total computation time till nowand if the thread is in L2 then total time denotes the total waiting time.

Added to thread.c -

1. static struct list ready_list_2 - This is the new queue for multi level scheduling. This is the second level queue
2. static long long clock - This is used for tracking global ticks and printing debugging information.

Functions that are added/modified

#define added -

1. #define which_ready_list(t) (((t)->qno) ? (&ready_list_2) : (&ready_list)) - This is used to find out the current queue or ready list the thread is currently in.

No new functions are added. Functions that are modified include:

1. void thread_init (void) - This was modified to initialise the new ready list and set the global clock to zero rest is left unchanged.
2. void thread_tick (void) - This function is modified to implement multi level scheduling. It increments the running time of the current running thread and performs context switch depending on the time quanta offered. It increments the total waiting time for all the threads in level 2 and increments the total computation time for the running thread in level 1 both of which are stored in the total_time field in the struct of thread and switching between levels is performed accordingly.
3. void thread_unblock (struct thread *t) - Will push a thread to level 1 or level 2 queue if it belonged to level 1 or level 2 queue before blocking.
4. tid_t thread_create (const char *name, int priority, thread_func *function, void *aux) - only some print statements are added. Rest of the function remains the same.
5. void thread_unblock (struct thread *t) - Will push a running thread to level 1 or level 2 queue depending on it's qno stored in the struct of thread and change the status to ready.
6. static struct thread *next_thread_to_run (void) - If level 1 queue is empty, then thread from L2 queue is popped else a thread from level 1 queue is popped. If both the queues are empty idle_thread is scheduled.
7. static void init_thread (struct thread *t, const char *name, int priority) - changed the initialization of the newly added struct thread fields.

Files that are modified

Two files are modified:

1. thread.h - changed the fields in struct thread
2. thread.c - modified the above functions and added some data structures.

Buddy Memory Allocation

Existing Working of the Within page memory allocation

When a request for a particular number of bytes is made, it is rounded to the nearest power of 2 and assigned to the descriptor that manages block of that size.
If the request is for more than 2 KB, the request can't be fulfilled by 1 page hence multiple pages need to be allocated which is done by rounding the required pages to the nearest integer and then calling palloc_get_multiple() function.
When the requested bytes is less than 2KB, the requirement is rounded to the nearest block size that can fulfill the requirement, a block of that block size is searched in the free block list and if found the block is returned. If no block of the required size exists in the free block list a new page is allocated and the new page is split/broken into the blocks of the required block size and added to the free block list and then the new block of the required size is returned.

Data Structures that are added/modified

- Struct desc:
  - Removed size_t blocks_per_arena as it is irrelevant.
- Struct arena:
  - Removed size_t free_cnt.
  - Added struct list_elem elem for keeping list of pages allocated.
  - Added int arr[128] for keeping size of each successful malloc call made at each possible starting address.
- Added struct list page_list for keeping list of allocated pages

Functions that are added/modified

- void printMemory(void): This function prints number of pages, and then for each page it prints the blocks that are free in this page in

ascending order of addresses grouped by their sizes. It uses page_list and iterates over free_list in descs[] which is first sorted.

- bool cmp_addr(const struct list_elem *a, const struct list_elem *b, void *aux): Comparator function for sorting a list. Return true if address of a < address of b.
- void malloc_init (void): Changed maximum block size to PGSIZE / 2 and removed initialization of removed variables.
- void *malloc(size_t size): It first asserts that size is no more than PGSIZE/2. Then it proceeds to find smallest free block size of which is no smaller than size. If none is found, then a new page is allocated using palloc_get_page, its arena is initialized(set magic, memset arr to 0) and the corresponding 2048 byte block is pushed in its corresponding free list. Then we split this block into 2 smaller blocks of equal sizes and the one with larger address is marked as free and the one with smaller address is considered till out size fits in this smaller block. This stops at 16 bytes as this is the smallest block size. Also, arr of the corresponding arena is set to mark the size on the address returned.
- static size_t block_size (void *block): Returns the number of bytes allocated for block using its arena's arr[].
- static struct arena * block_to_arena (struct block *b): Removed asserts with removed variables.
- static struct block * arena_to_block (struct arena *a, size_t idx): Removed the function.
- void free(void *p): The block is memset to 0xcc for debugging purposes if the corresponding #define is done. Corresponding entry in arr[] is set to 0 as this block is no more allocated. While this block's size is less than PGSIZE/2, It finds its buddy by xoring its address with its size(after taking appropriate offset from end of arena). It checks if the buddy is free by iterating over corresponding arr[] elements in it. If the buddy is not free, it breaks. Else, both of these are merged into one free block. Corresponding free lists are updated

and this part is repeated. If we come upon a free block of size PGSIZE/2 the corresponding is freed using palloc_free_page.

Files that are modified:

- malloc.h - added declaration for the function PrintMemory().
- malloc.c - added/modified all the functions and data structures explained above.