pintOS 1 reports

1.

2.

//priority//

**Thread.c**

**Init\_thread**:

t->init\_priority = priority; (505)

**Thread\_yield**:

list\_insert\_ordered(&ready\_list, &curr->elem, priority\_scheduling, NULL); ()

**Thread\_unblock**:

list\_insert\_ordered(&ready\_list, &t->elem, priority\_scheduling, NULL); (250)

**bool priority\_scheduling** (const struct list\_elem \*a, const struct list\_elem \*b, void \*aux UNUSED)

{

struct thread \*ra = list\_entry(a, struct thread, elem);

struct thread \*rb = list\_entry(b, struct thread, elem);

return ra->priority > rb->priority;

} (78)

**Thread\_create**:

priority\_preemption (); (232)

**void priority\_preemption** (void)

{

struct thread \*cur = thread\_current();

struct thread \*t = list\_entry(list\_front(&ready\_list), struct thread, elem);

if (cur != idle\_thread)

{

if (t->priority > cur->priority)

thread\_yield();

}

} (85)

**Thread\_wakeup**:

priority\_preemption(); (373)

**Thread.h**

**bool priority\_scheduling** (const struct list\_elem \*a, const struct list\_elem \*b, void \*aux UNUSED); (145)

Thread struct:

**int init\_priority**;

//priority done//

//semaphore//

**Synch.h**

**bool sema\_priority\_scheduling**(const struct list\_elem \*a, const struct list\_elem \*b, void \*aux);

**struct semaphore\_elem**

{

struct list\_elem elem; /\* List element. \*/

struct semaphore semaphore; /\* This semaphore. \*/

};

**Synch.c**

**bool sema\_priority\_scheduling** (const struct list\_elem \*a, const struct list\_elem \*b, void \*aux UNUSED)

{

} (36)

**Sema\_down**:

list\_insert\_ordered (&sema->waiters, &thread\_current ()->elem, thread\_priority\_scheduling, NULL); (87)

**Sema\_up:**

If(intr\_context())

Priority\_preemption(); (135)

//synch done//

//donation//

**Thread.h**

**struct lock \*wait\_on\_lock;** (107)

**void donate\_priority**(void); (150)

**struct list donation\_list;** (110)

**Thread.c**

**struct list\_elem donation\_elem**; ()

**void donate\_priority**(void); (152)

**void refresh\_donate**(struct lock \*lock);

**synch.h**

**bool donate\_priority\_scheduling**(const struct list\_elem \*a, const struct list\_elem \*b void \*aux UNUSED)

struct thread

{

/\* Owned by thread.c. \*/

tid\_t tid; /\* Thread identifier. \*/

enum thread\_status status; /\* Thread state. \*/

char name[16]; /\* Name (for debugging purposes). \*/

uint8\_t \*stack; /\* Saved stack pointer. \*/

int priority; /\* Priority. \*/

int init\_priority; /\* Initial Priority \*/

struct list\_elem allelem; /\* List element for all threads list. \*/

/\* Shared between thread.c and synch.c. \*/

struct list\_elem elem; /\* List element. \*/

#ifdef USERPROG

/\* Owned by userprog/process.c. \*/

uint32\_t \*pagedir; /\* Page directory. \*/

#endif

/\* Owned by thread.c. \*/

unsigned magic; /\* Detects stack overflow. \*/

int64\_t wticks;

struct list donation\_list;

struct list\_elem donation\_elem;

};