# Implementation of virtual time in glibc

Variable added in ***sysdeps/x86/bits/pthreadtypes.h*** for mutex:

```
#  define __SIZEOF_PTHREAD_MUTEX_T 96
```

We need to change the size since we are going to add new variables in ***pthread_mutex_t*** structure.

```
struct vtime_mutex_node
{
  int tid;
  double  g_time_at_request;
  double lock_holder_v_time_at_request;
  TAILQ_ENTRY(vtime_mutex_node) tailq;
};
```

TAILQ will hold the waiting task waiting for the mutex lock

```
TAILQ_HEAD(vtime_mutexq, vtime_mutex_node);

   double lock_acquisition_g_time;
    double lock_acquisition_v_time;
    //__v_t_list waiter_list;
    struct vtime_mutexq waiter_list;
    volatile  int vt_lock;
```

Initialize the mutex related variables in ***nptl/pthread_mutex_lock_init.c***:

```
pthread_spin_init(&(mutex->__data.vt_lock), PTHREAD_PROCESS_SHARED);
TAILQ_INIT(&mutex->__data.waiter_list);
mutex->__data.lock_acquisition_v_time = 0;
mutex->__data.lock_acquisition_g_time = 0;
```

Update the mutex related variables in ***nptl/pthread_mutex_lock.c***:

```
static double
g_time_ms(void){
                                        struct timeval g_time;
                                        gettimeofday(&g_time , NULL);
                                        double g_ms = (double)g_time.tv_sec*1000000 +
                                    (double)g_time.tv_usec;
                                        return g_ms;
                                    }
```

Function to enqueue waiting task for this mutex:

```
static void
ahmed_enqueue_tid(pthread_mutex_t
*mutex, int tid){
                                                //printf("Enter enque!\n");
                                                __v_t_list *new_node =
                                            (__v_t_list*)malloc(sizeof(__v_t_list));
                                                new_node->tid = tid;
                                                new_node->g_time_at_request = g_time_ms();
                                            //
                                            printf("***************owner:%d\n",mutex->__data.__own
                                            er);
                                                if (mutex->__data.__owner!=0){// already locked
                                                        new_node->lock_holder_v_time_at_request
                                            = (double)syscall(333,mutex->__data.__owner);
                                                }
                                                new_node->next =
                                            mutex->__data.waiter_list.next;
                                                new_node->prev = &mutex->__data.waiter_list ;


                                            //      printf("^^^^^^^^^^^^^^^^^^Requesting for
                                            lock.%d\n",mutex->__data.vt_lock);
                                            //      pthread_spin_lock(&mutex->__data.vt_lock);
```

```
//      printf("^^^^^^^^^^^^^^^^^^^^^^^^gets
lock^^^^^^^^^^^^^^^^^^");
//      pthread_spin_unlock(&mutex->__data.vt_lock);
        if(new_node->next != NULL){//Has previous item
                new_node->next->prev = new_node;
        }
        mutex->__data.waiter_list.next = new_node;
//      pthread_spin_unlock(&mutex->__data.vt_lock);
}
```

Function to dequeue task which is no more waiting for the mutex:

```
static void dequeue_tid(pthread_mutex_t
*mutex, int tid){
        //printf("Dequeue called with %d\n",tid);

        __v_t_list *temp =
&mutex->__data.waiter_list;
        __v_t_list  *node_2b_deleted;
        //
pthread_spin_lock(&mutex->__data.vt_lock);
                while(temp->next !=NULL){
                if(temp->next->tid == tid){
                        node_2b_deleted =
temp->next;
                //      printf("Found match in
dequeue %d\n",tid);
                        if(temp->next->next!=
NULL){
                                //printf("enter if
!");

        temp->next->next->prev = temp;
                                //printf("if
executed!\n");
```

```
                        }
                        temp->next =
temp->next->next;
                        //printf("temp next
executed!\n");
                        free(node_2b_deleted);
                        break;
                    }
                    temp = temp->next;
                }
//
pthread_spin_unlock(&mutex->__data.vt_lock);


}
```

Add the lock holding time of the current task as waiting time for other task waiting in the queue in ***nptl/pthread_mutex_unlock.c***:

```
static void
add_vtime(pthr
ead_mutex_t
*mutex){
                    __v_t_list *temp = &mutex->__data.waiter_list;
                     double delta_v = 0;
                      while(temp->next !=NULL){
                            if(mutex->__data.lock_acquisition_g_time >=
temp->next->g_time_at_request){
                                  delta_v =
(double)syscall(333,mutex->__data.__owner) -
mutex->__data.lock_acquisition_v_time;
                                  //printf("**In unlock**\towner:%d\trequest time is
smaller, delta_v:%lf\t lock_accquisition_v_time:%lf
\n",mutex->__data.__owner,delta_v/10e8,mutex->__data.lock_acquisition_v_t
ime/10e8);
                            }
                            else{
```

```
                        delta_v =
(double)syscall(333,mutex->__data.__owner) -
temp->next->lock_holder_v_time_at_request;
                //      printf("Request time is larger, delta_v:
%lld\n",delta_v);
                }

                double elapsed_time_since_last_vtime_update =g_time_ms() -
(double)syscall(337,mutex->__data.__owner)/1000;
                delta_v += elapsed_time_since_last_vtime_update;

                printf("*In Unloack*\tadding %lf to thread: %d, elaspsed
time since last vtime update %lf\n",delta_v,
temp->next->tid,elapsed_time_since_last_vtime_update);
                syscall(334,temp->next->tid,(long long int)delta_v);

                 temp = temp->next;

        }

}
```

Variable added in ***sysdeps/nptl/internaltypes.h*** for semaphore:

```
typedef struct vtime_node
{
  int tid;
  unsigned long *times_at_request;
  struct vtime_node *prev;
  struct vtime_node *next;

} vtime_node;

  int latest_post_from;
  double latest_time_at_posting;
  pthread_spinlock_t lock;
  vtime_node  waiter_list;
```

Modification in ***nptl/sem_wait.c***

```
static void
enqueue_tid(struct
new_sem* semaphore, int
tid){

        int expected_num_thread = 20;
        vtime_node *new_node =
(vtime_node*)malloc(sizeof(vtime_node));
        new_node->tid = tid;

        new_node->times_at_request = (unsigned long
*)malloc((expected_num_thread+1)*3*sizeof(unsigned long));
        syscall(339,expected_num_thread,
new_node->times_at_request, tid);

        new_node->next = semaphore->waiter_list.next;
        new_node->prev = &semaphore->waiter_list ;
        if(new_node->next != NULL){//Has previous item
                new_node->next->prev = new_node;
        }
        semaphore->waiter_list.next = new_node;

}

static void add_waiting_time(struct new_sem* semaphore, int
tid){
//printf("Dequeue called with %d\n",tid);

        //remove node from link_list
```

```
        vtime_node *temp = &semaphore->waiter_list;
        vtime_node *node_2b_deleted = NULL;
                while(temp->next !=NULL){
               if(temp->next->tid == tid){
                       node_2b_deleted = temp->next;
                //      printf("Found match in dequeue
%d\n",tid);
                        if(temp->next->next!= NULL){ // more
nodes after it
                                //printf("enter if !");


                                temp->next->next->prev = temp;
                                //printf("if executed!\n");
                        }
                        temp->next = temp->next->next;
                        //printf("temp next executed!\n");
                        break;
                }
                temp = temp->next;
        }


        //Add waiting time to the waiter
        int i;
        double waiting_time = 0 ;
        for(i= 1; i<node_2b_deleted->times_at_request[0]; i++){
// 0th index contain the number of thread running at the wait
call by this thread
                if
(node_2b_deleted->times_at_request[3*i]==semaphore->latest_post
_from){
                        waiting_time =
semaphore->latest_time_at_posting -
node_2b_deleted->times_at_request[3*i+1];
```

```
                                        break;
                                    }


                                }
                                free(node_2b_deleted);
                                printf("waiting time for
                            semaphore:%lf\n",waiting_time);
                                syscall(334, tid, (long long int)waiting_time);



                            }
```

Modification in nptl/sem_post.c:

```
isem->latest_post_from = (int)
syscall(__NR_gettid);
                                    isem->latest_time_at_posting =
                                   (double)syscall(333,isem->latest_post_from);
```

Variable added in ***sysdeps/nptl/internaltypes.h*** for barrier:

```
double grp_mx_vtime;
```

Initialize the barrier related variables in ***nptl/pthread_barrier_init.c***:

```
ibarrier->grp_mx_vtime = 0;
```

Adding waiting time at barrier in ***nptl/pthread_barrier_wait.c***:

```
  double vtime =
  (double)syscall(333,self
  _pid);
                              //  printf("in barrier wat: %d\tv_time returned %lf\t
                              grp_mx_vtime %lf\n",self_pid, vtime, bar->grp_mx_vtime);
                               bar->grp_mx_vtime = bar->grp_mx_vtime >
                              vtime?bar->grp_mx_vtime:vtime;


syscall(338,self_pid,(long long int)bar->grp_mx_vtime);
```

Variable added in ***nptl/pthread_join.c*** for thread join:

```
        int tid = pd->tid;
        syscall(340,tid);
```