

Sistemas Operativos

Implementación de mensajes
En nThread

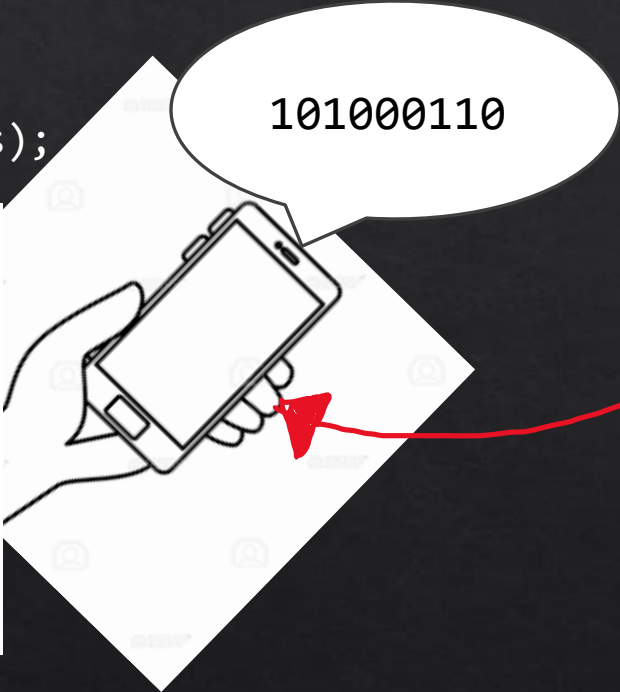
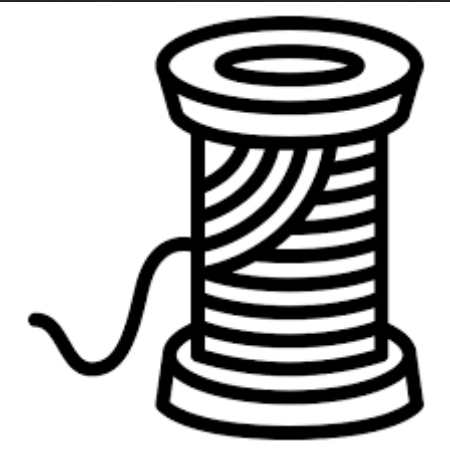
Pablo Jaramillo

Diapositivas basadas en las de
Diego Madariaga (2022-2)



Mensajes

```
nReceive(*pth, timeout_ms);
```



101000110



```
nSend(th, *msg);
```



Mensajes

Como funcionan

```
int nSend(nThread th, void *msg)
```

Envía un mensaje `msg` al thread `th`.
Suspende al thread hasta que se reciba una respuesta desde `th`.

Retorna el valor recibido.

```
void nReply(nThread th, int rc)
```

Envía una respuesta con `rc` a un mensaje recibido de `th`.

No suspende al thread.

```
void *nReceive(nThread *pth, int timeout_ms)
```

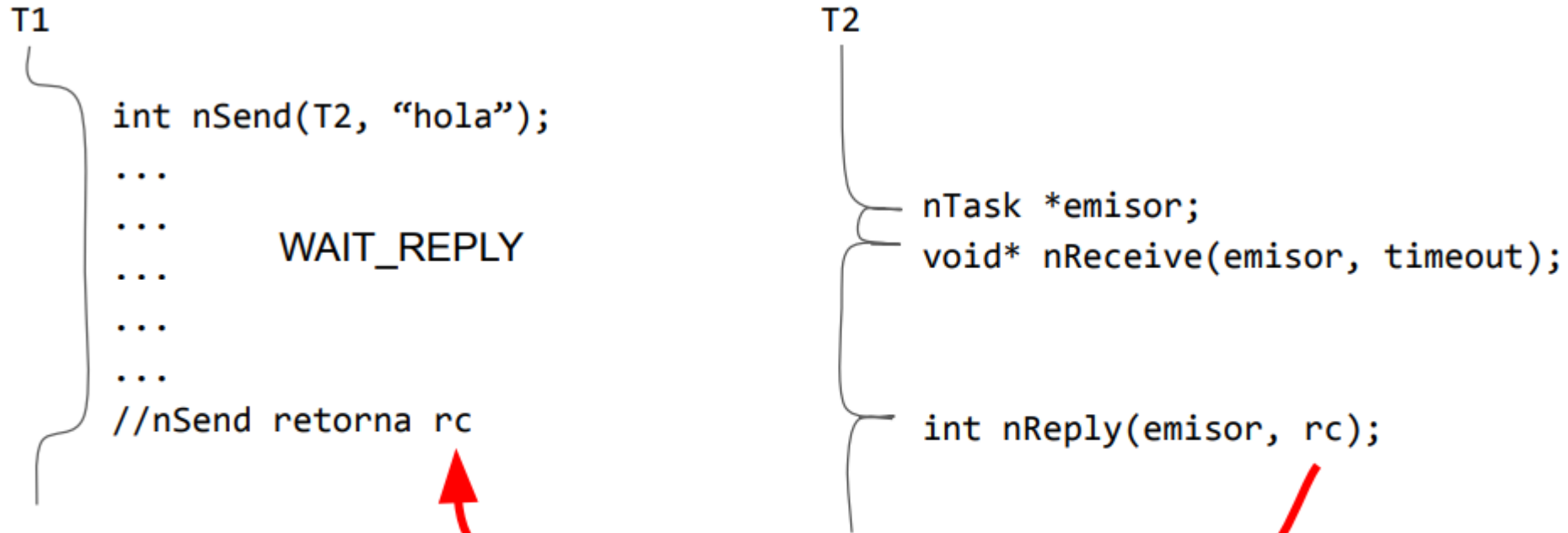
Suspende al thread hasta recibir un mensaje. Registra al thread desde donde viene el mensaje en el puntero `*pth`. Retorna el mensaje recibido.

La suspensión dura `timeout_ms`, si no se recibe un mensaje retorna de todas formas, si este valor es 0 entonces no se suspende.

Si es `<0` la suspensión es infinita hasta recibir un mensaje.

Mensajes

Como funcionan



nSend esperará hasta recibir una respuesta del thread que utiliza nReceive.

Mensajes

Como funcionan

T1

```
int nSend(T2, "hola");  
...  
...    WAIT_REPLY  
...  
//nSend retorna rc
```

T2

```
nTask *emisor;  
void* nReceive(emisor, timeout);  
...  
...    WAIT_SEND / WAIT_SEND_TIMEOUT  
...  
int nReply(emisor, rc);
```

nReceive
esperará hasta
recibir un
mensaje desde
nSend.

nSend esperará
la respuesta para
confirmar que su
mensaje fue
recibido.



Ejemplo de uso de mensajes:
Impresora Compartida

Impresora compartida

Comparación

Patrón Request

```
void obtenerImpresora(){
    pthread_mutex_lock(&m);
    Request req = {FALSE, PTHREAD_COND_INITIALIZER};
    put(q, &req);
    pthread_cond_signal(&obtener);
    while (!req.ready){
        pthread_cond_wait(&req.w, &m);
    }
    pthread_mutex_unlock(&m);
}

void devolverImpresora(){
    pthread_mutex_lock(&m);
    ocupada = FALSE;
    pthread_cond_signal(&devolver);
    pthread_mutex_unlock(&m);
}
```

✧ Mensajes ✧

```
enum Mensaje {OBTENER, DEVOLVER};

void obtenerImpresora(){
    int msg = OBTENER;
    // Bloquear thread hasta recibir respuesta
    nSend(impresora, &msg);
}

void devolverImpresora(){
    int msg = DEVOLVER;
    nSend(impresora, &msg);
}
```

Impresora compartida

Comparación

Patrón Request

```
void ImpresoraServer(){
    while(TRUE){
        pthread_mutex_lock(&m);
        if(emptyQueue(q)){
            struct timespec ts;
            clock_gettime(CLOCK_REALTIME, &ts);
            ts.tv_sec += 60*5;
            while(emptyQueue(q)
                && pthread_cond_timed_wait(&obtener, &m, &ts)
                != ETIMEDOUT
            ){
                //wait por 5 mins
            }
            if(emptyQueue(q)){
                modoBajoConsumo();
                while (emptyQueue(q)){
                    pthread_cond_wait(&obtener, &m);
                }
                modoUsoNormal();
            }
        }
        if(!emptyQueue(q)){
            Request *req = get(q);
            req -> ready = TRUE;
            ocupada = TRUE;
            pthread_cond_signal(&req->w);
        }
        while(ocupada){
            pthread_cond_wait(&devolver, &m);
        }
        pthread_mutex_unlock(&m);
    }
}
```


Impresora compartida

Comparación

✧ Mensajes ✧

```
int impresoraServer(void *_ignored){
    Queue *q = makeQueue();
    int ocupado = FALSE;

    nThread t;
    int *msg;

    while (TRUE){
        // Si no esta ocupada, esperar
        if(!ocupado){
            // esperar 5 minutos (5min * 60s * 1000ms)
            msg = (int *) nReceive(&t, 60 * 5 * 1000);
            if(t == NULL){
                modoBajoConsumo();
                msg = (int*) nRecieve(&t, -1); // esperar
                modoUsoNormal();
            }
        } else { // Esperar a que se desocupe impresora
            msg = (int*) nReceive(&t, -1); // esperar
        }
    }
}

if(*msg == OBTENER){
    // Encolar si esta ocupada,
    if(ocupado)
        put(q,t);
    else{ // responder si esta disponible
        ocupado = TRUE;
        nReply(t,0);
    }
} else if( *msg == DEVOLVER){
    nReply(t, 0); // "ok, la devolviste"
    if( EmptyFifoQueue(q)){
        ocupado = FALSE;
    } else {
        // Entregar impresora a siguiente thread
        nThread *t2 = (nThread*) get(q);
        nReply(t2, 0);
    }
}
}
```

1.- Implementar mensajes

Implementar Mensajes

Implemente:

```
int nSend(nThread th, void *msg)
```

```
void *nReceive(nThread *pth, int timeout_ms)
```

```
void nReply(nThread th, int rc)
```