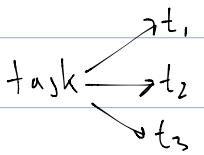
thread? not a process — independent unit running concurrently in a process

(1) PC (program Counter)
(2) stack
(3) registers

} allocated to each

(1) specialist paradigm



C(I) = A(I) + B(I), I = 1..100

C1: 1..20
C2: 21..40
C3: 41..60
C4: 61..80
C5: 81..100

SIMD Model
single instruction multiple data string

(2) client-server paradigm

client 1 → server 1

client 2 → server 3

client 3 → server 2

word processor

user input thread

spelling or grammar checking thread

document layout thread

etc.

(3) assembly line paradigm

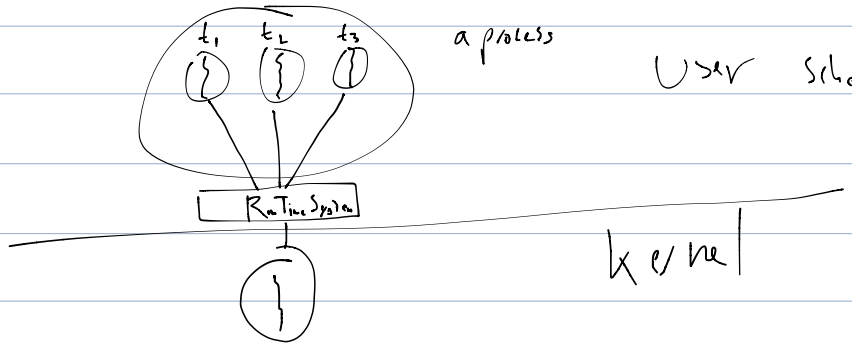
a task → thread 1 → thread 2 → thread 3

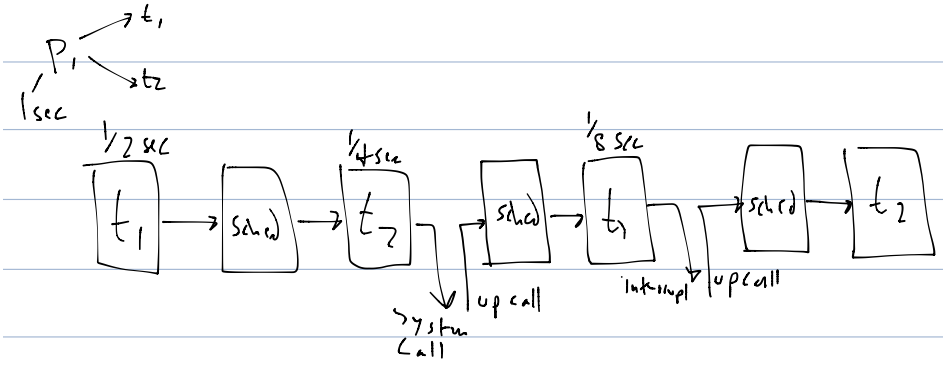
compiler (2 passes)

pass 1: Symbol table

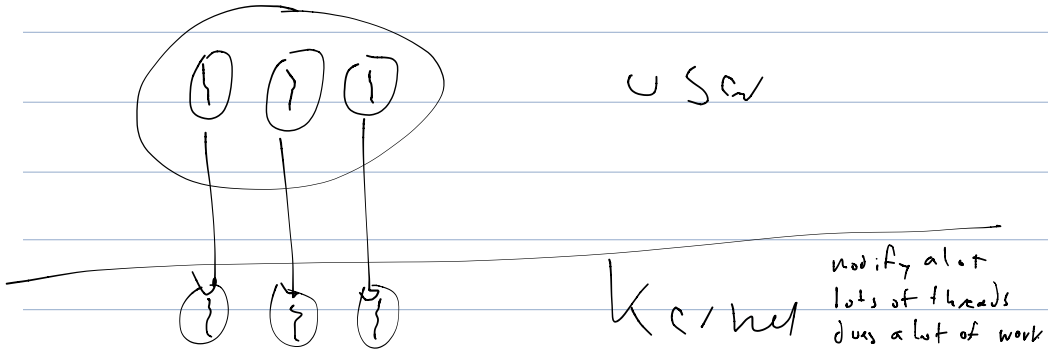
pass 2: object code

(1) many-to-one approach (multiple threads on 1 kernel thread) (Solaris)

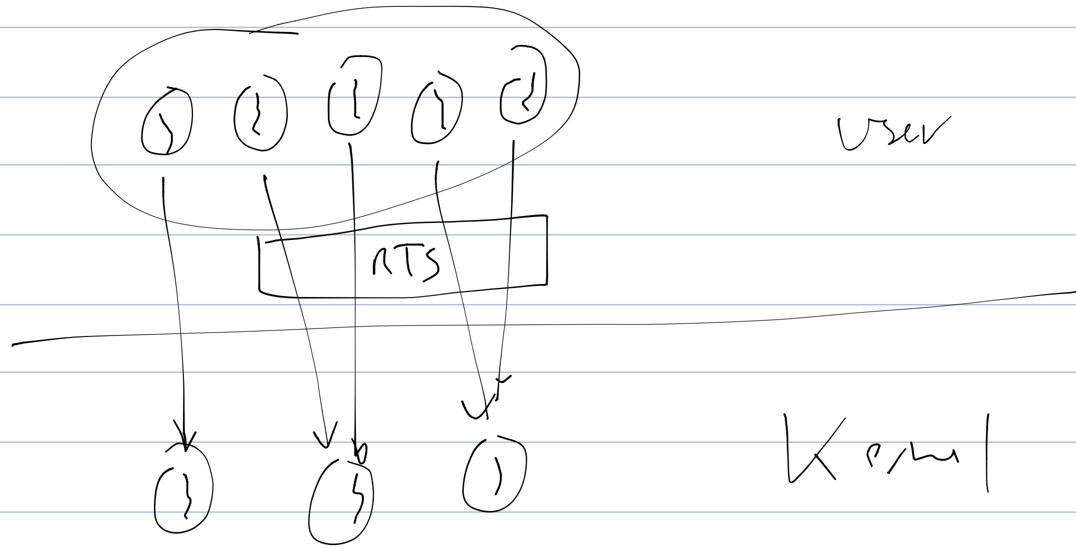




(2) One-to-one (Linux, Ulix, Windows)



(3) many-to-many




```
int sum;                                              sum 5

int main(int argc, char* argv[]) {          (t1)
                      // <- global variable avec
    pthread_t tid; //tid[len]
    pthread_attr_t attr;        // attribute of that type
    pthread_attr_init(&attr);   // default values
                                // (t2) -> name of fcn which will become a thread
    pthread_create(&tid, &attr, runner, argv[1]);
                                // &status
    pthread_join(tid, NULL);    // <- block the parent (w/out = error)
    printf("sum = %u", sum);
}


void *runner(void *param)
  {

    int upper = atoi(param);
    sum=0;
    for(int i=0; i<= upper; i++)
        sum+=i;
```

```
    pthread_exit(0); ← (w/out = fine, garbage collection, but takes time)
}
```

```
for(int=0; i< len; i++)
    pthread_create(&tid[i], NULL, runner, NULL);
```

```
for(int i=0; i< len; i++)
    pthread_join(tid[i], NULL);
```

P1:	P2:	(1) shared resource (file, printer, etc)
while(1) {	while(1) {	(2) if executed at assembly level
⋮	⋮	
counter++; (3)	counter--;	$P_1 \to P_2$: 3
⋮	⋮	$P_2 \to P_1$: 3
}	}	interleaving?
(1) R1 ← counter;	(1)' R2 ← counter;	(1)', (2)', (1), (2), (3)', (3)
(2) R1 ← R1+1;	(2)' R2 ← R-1;	R2=3 R2=2 R1=3 R1=4 C=2 C=4
(3) counter ← R1;	(3)' counter ← R2;	(3), (3)'
		C=4 C=2

ready list must be locked to access critical section = 1 at a time for thread process (TP)

Entry
Critical
exit
remainder
} section