Processos e Threads

Mutex locks recursivos e com verificação de erros

Deadlock de uma thread só

```
void f() {
  mutex_lock(&lock);
  mutex_lock(&lock);
}
```

Veja o código: deadlock.c

Locks simples

Registro controlado por um lock

```
struct registro {
   mutex_t lock_reg;
   char nome[30];
   float saldo_conta_corrente;
   float saldo_poupanca;
} Reg;
```

- Como escrever rotinas de impressão?

Locks recursivos

Rotinas de impressão

```
void imprime_saldo_conta_corrente(Reg* reg);
void imprime_saldo_poupanca(Reg* reg);
void imprime_completo(Reg* reg);
```

Locks simples

Rotinas de impressão

```
void imprime_saldo_conta_corrente(Reg* reg) {
   mutex_lock(reg->lock);
   printf("Saldo conta corrente: %f",
           reg->saldo_conta_corrente);
   mutex_unlock(reg->lock);
}

void imprime_saldo_poupanca(Reg* reg) {
   /* ... */
}
```

Locks simples

Rotinas de impressão

```
void imprime_completo(Reg* reg) {
   mutex_lock(reg->lock);
   printf("Nome: %s", reg->nome);
   printf("Saldo conta corrente: %f",
          reg->saldo_conta_corrente);
   printf("Saldo poupanca: %f",
          reg->saldo_poupanca);
   mutex_unlock(reg->lock);
}
```

- Replicação de código—se quisermos alterar a formatação das strings temos de mexer em dois lugares. :-(

Locks simples

Rotinas de impressão

```
/* Função não atômica */
void aux_imprime_saldo_conta_corrente(Reg* reg) {
   printf("Saldo conta corrente: %f",
           reg->saldo_conta_corrente);
}
/* Função atômica */
void imprime_saldo_conta_corrente(Reg* reg) {
   mutex_lock(reg->lock);
   aux_imprime_saldo_conta_corrente(reg);
   mutex_unlock(reg->lock);
}
```

Locks simples

Rotinas de impressão

```
void imprime_completo(Reg* reg) {
   mutex_lock(reg->lock);
   printf("Nome: %s", reg->nome);
   aux_imprime_saldo_conta_corrente(reg);
   aux_imprime_saldo_conta_poupanca(reg);
   mutex_unlock(reg->lock);
}
```

- Aumento desnecessário do número de funções.
- Risco de uma função não atômica ser invocada indevidamente.

Locks recursivos

Rotinas de impressão

```
void imprime_completo(Reg* reg) {
   mutex_lock(reg->lock);
   printf("Nome: %s", reg->nome);
   imprime_saldo_conta_corrente(reg);
   imprime_saldo_poupanca(reg);
   mutex_unlock(reg->lock);
}
```

Locks recursivos

```
void f() {
  mutex_lock(&lock);
  /* faz alguma coisa */
  mutex_unlock(&lock);
}

void g() {
  mutex_lock(&lock);
  f();
  /* faz outra coisa */
  mutex_unlock(&lock);
}
```

Locks recursivos

Implementação a partir de locks simples e variáveis de condição

```
typedef struct {
  pthread_t thr;
  cond_t cond;
  mutex_t lock_var;
  int c;
} rec_mutex_t;
```

rec_mutex_lock()

```
int rec_mutex_lock(rec_mutex_t *rec_m) {
  pthread_mutex_lock(&rec_m->lock_var);
  if (rec_m->c == 0) {  /* Lock livre */
    rec_m->c = 1;
    rec_m->thr = pthread_self();
  } else  /* Mesma thread */
    if (pthread_equal(rec_m->thr,
                      pthread_self()))
        rec_m->c++;
    else
     /* Thread deve esperar */
```

rec_mutex_lock()

```
  else {
     /* Thread deve esperar */
     while (rec_m->c != 0)
       pthread_cond_wait(&rec_m->cond,
                         &rec_m->lock_var);
     rec_m->thr = pthread_self();
     rec_m->c = 1;
  }
  pthread_mutex_unlock(&rec_m->lock_var);
  return 0;
}
```

rec_mutex_unlock()

```
int rec_mutex_unlock(rec_mutex_t *rec_m) {
  pthread_mutex_lock(&rec_m->lock_var);
  rec_m->c--;
  if (rec_m->c == 0)
    pthread_cond_signal(&rec_m->cond);
  pthread_mutex_unlock(&rec_m->lock_var);
  return 0;
}
```

Verificação de erros

rec_mutex_unlock()

```
int rec_mutex_unlock(rec_mutex_t *rec_m) {
  pthread_mutex_lock(&rec_m->lock_var);
  if (!pthread_equal(rec_m->thr,
                     pthread_self())) {
      pthread_mutex_unlock(&rec_m->lock_var);
      return ERROR;
  }
  else
    /* ... */
```