EXAMEN SISTEMAS OPERATIVOS AVANZADOS TEORIA

VIERNES 13 DICIMEBRE 2019

NOMBRE:

1. Debajo se muestran los patrones de acceso a memoria de tres procesos en un sistema que usa registro *base* y *bounds* para hacer la traducción entre VA y PA. Completar los valores no conocidos:

VA	**PA**	**VA**	**PA**	**VA**	**PA**
0	1000	100	3400	0	¿ ?
100	1100	2000	5300	100	¿ ?
1999	2999	2001	¿ ?	2000	¿ ?
2000	[fault]	3000	6300	2001	[fault]
BASE: ?	**BOUNDS: ?**	**BASE: ?**	**BOUNDS: ?**	**BASE:6050**	**BOUNDS: ?**

2. Algunos de las siguientes combinaciones de *threads* y locks pueden dar lugar a *deadlock*. ¿Cuáles no y por qué?
   a) Thread1, adquiere lock1 y lock2 en un orden arbitrario. Thread2 adquiere siembre lock1 antes que lock2
   b) Thread1, adquiere siempre lock1 antes que lock2. Thread2 adquiere siembre lock1 antes que lock2
   c) Thread1, adquiere lock1 y lock2 en un orden arbitrario. Thread2 adquiere siembre lock2 después lock3.
   d) Thread1, adquiere lock1 y después lock2. Thread2 adquiere siembre lock2 después lock3. Thread3 adquiere siembre lock1 después lock3.
   e) Thread1, adquiere lock1 y después lock2. Thread2 adquiere siembre lock2 después lock3. Thread3 adquiere lock1 y lock2 en un orden arbitrario.

3. Reconstruir la tabla de páginas (de un solo nivel, suponiendo que el PTE tiene PFN y bit de validez o presencia) de un proceso a partir de la secuencia de traducciones es la siguiente

VA	**PA**
0x1063	0x2063
0x67b4	0x67b4
0x584a	0xe84a
0x4dfe	Inválida
0x388a	Inválida
0x1c6b	0x2c6b
0x50a9	0xe0a9
0x0bc6	Inválida
0x2a9f	0x9a9f
0x742b	Inválida
0x4b5e	Invalida
0x5597	0xe597

Detalles
- El espacio de direcciones virtual 32KB
- Tamaño de página 4KB
- Tamaño de la memoria física 64KB

4. Suponer el siguiente código, el cual desafortunadamente tiene un bug. Suponer que las llamadas pthread create() y calloc() siempre funcionan y que una dereferencia a NULL produce un fallo de segmentación siempre.

```
void *background_malloc(void *arg) {
  int **int_ptr = (int **) arg;
  *int_ptr = calloc(1, sizeof(int));
  **int_ptr = 10;
  return NULL;
}

int main(int argc, char *argv[]) {
  pthread_t p1;
  int *result = NULL;
  pthread_create(&p1, NULL, background_malloc, &result);
  printf("%d\n", *result);
  return 0;
}
```

Que situaciones de las siguientes son posible y cuales no (justificando la respuesta)

a) El código imprime 0
b) El código imprime 10
c) El código imprime 100
d) Se produce un fallo de segmentación
e) El código se cuelga indefinidamente

5. Cuales de las siguientes afirmaciones sobre tablas de páginas multinivel (TPM) son correctas y cuales no (justificando la respuesta)
   a) Una TPM puede requerir más páginas que una tabla de paginas lineal (TPL)
   b) Es más fácil reservar los marcos a emplear por una TPM que por una TPL
   c) LA búsqueda en una TPM es más costosa que en na TPL
   d) Cuanto mayor es el espacio de direcciones virtual mayor es el número de niveles en una TPM
   e) El TLB permite hacer las TPM aún más pequeñas
   f) Con una TPM los fallos de TLB son menos costosos
   g) Con una TPM los fallos de página son menos costosos

6. Suponer el siguiente código

```
void *printer(void *arg) {
  char *p = (char *) arg;
  printf("%c", *p);
  return NULL;
}

int main(int argc, char *argv[]) {
  pthread_t p[5];
  for (int i = 0; i < 5; i++) {
    char *c = malloc(sizeof(char));
    *c = 'a' + i; // hint: 'a' + 1 = 'b', etc.
    pthread_create(&p[i], NULL, printer, (void *) c);
  }
  for (int i = 0; i < 5; i++)
    pthread_join(p[i], NULL);
  return 0;
}
```

Suponiendo que todas las llamadas al sistema o de librería funcionan correctamente, que situaciones de las siguientes son posibles y cuales no (justificando la respuesta).

a) El código imprime abcd
b) El código imprime edcba
c) El código imprime cccde
d) El código imprime eeeee
e) El código imprime aaaaa

EXAMEN SISTEMAS OPERATIVOS AVANZADOS PRACTICAS

VIERNES 13 DICIMEBRE 2019

NOMBRE Y FIRMA: USUARIO GITLAB:

Sobre la base de tu implementación de la práctica 3 de la asignatura, crear un nuevo proyecto llamado Examen2 en gitlab que incluya como *developer* a vpuente@gmail.com y que en dos *ramas (*con nombre *error y wait)* separadas incorpore las siguientes modificaciones.

La llamada *shmem_access* debe separar la posibilidad de que el acceso sea en modo escritura o lectura. Se deberá agregar un segundo argumento a la llamada indicando el modo pretendido. Con el fin de mantener la coherencia, deberemos prevenir que varios procesos estén concurrentemente accediendo a la misma página si uno de ellos ha accedido modo escritura (es decir, se permiten múltiples lectores concurrentes por página, pero solo un "escribidor"). Cuando las condiciones no se cumplan:

1. Branch1 (***error***) La llamada al sistema debe devolver un error (6 pt). *Git hash:*_____

2. Branch2 (***wait***) La llamada al sistema se debe bloquear hasta que la condición de acceso se cumpla (4pt). *Git hash:*____

En ambos casos es necesario escribir un programa de prueba que verifique la funcionalidad.