Process/Thread. Circle one or both of P or T, as applies.

1. P / T Has an associated control block.

2. P / T Has an associated data segment.

3. P / T Has an associated SP (stack pointer).

4. P / T Has an associated PC (program counter).

5. P / T Is created in response to the UNIX system call fork().

6. P / T The abstraction for protection provided by the OS kernel.

7. P / T A single execution sequence that represents a separately schedulable task.

8. Are threads considered more efficient than processes? If so, why? If not, why not?

Yes, threads are more efficient than processes, as they are cheaper to create/destroy, share the same memory map, faster to switch between other threads than between processes, and they communicate between other threads using loads and stores instead of sends/receives.

9. In the following thread state diagram, label the states (i.e., fill in the blanks above or below the circles) with the state names: Finished, Init, Ready, Running, and Waiting.

1. \_\_Init\_\_\_\_ 2. \_\_\_Ready\_\_ 3. \_Running\_ 4. \_Finished\_\_

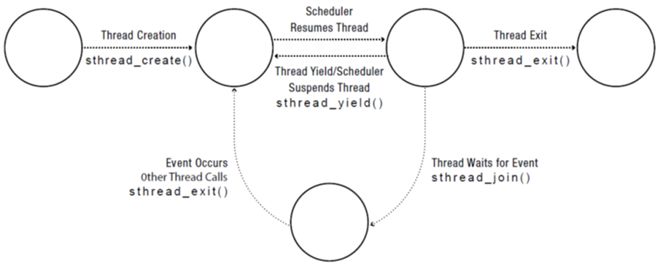
2

4

5

3

1



5. \_\_Waiting\_\_\_

10. Consider the following program using the thread library from ospp.cs.washington.edu.

/\* create.c -- Example multi-threaded program

\*

\* Compile with

\* > gcc -g -Wall -Werror -D\_POSIX\_THREAD\_SEMANTICS create.c -c -o create.o

\* > gcc -g -Wall -Werror -D\_POSIX\_THREAD\_SEMANTICS thread.c -c -o thread.o

\* > gcc -pthread create.o thread.o -o create

\* Run with

\* > ./create

\* or

\* > valgrind --tool=helgrind ./create

\*/

#include <stdio.h>

#include <string.h>

#include <assert.h>

#include "thread.h"

#define NTHREADS 2

void \* go (void \*vp);

double x;

int main(int argc, char \*\*argv) {

int i;

thread\_t threads[NTHREADS];

for (i = 0; i < NTHREADS; i++) {

thread\_create\_p(&(threads[i]), &go, NULL);

}

for (i = 0; i < NTHREADS; i++) {

thread\_join(threads[i]);

}

return 0;

}

void \* go (void \*vp) {

double y;

printf( "address of x is %p, address of y is %p\n", &x, &y );

thread\_exit(0);

// Not reached

return NULL;

}

1. How many different copies of the variable x are there?

1

1. How many different copies of the variable y are there?

2