z/OS UNIX System Services Daemons and Servers

RUGONE
October 17, 2013

Bruce R. Wells
brwells@us.ibm.com

Trademarks

- See url http://www.ibm.com/legal/copytrade.shtml for a list of trademarks.

Agenda

- Daemons
- Servers

- (Were you expecting something else?)

Background

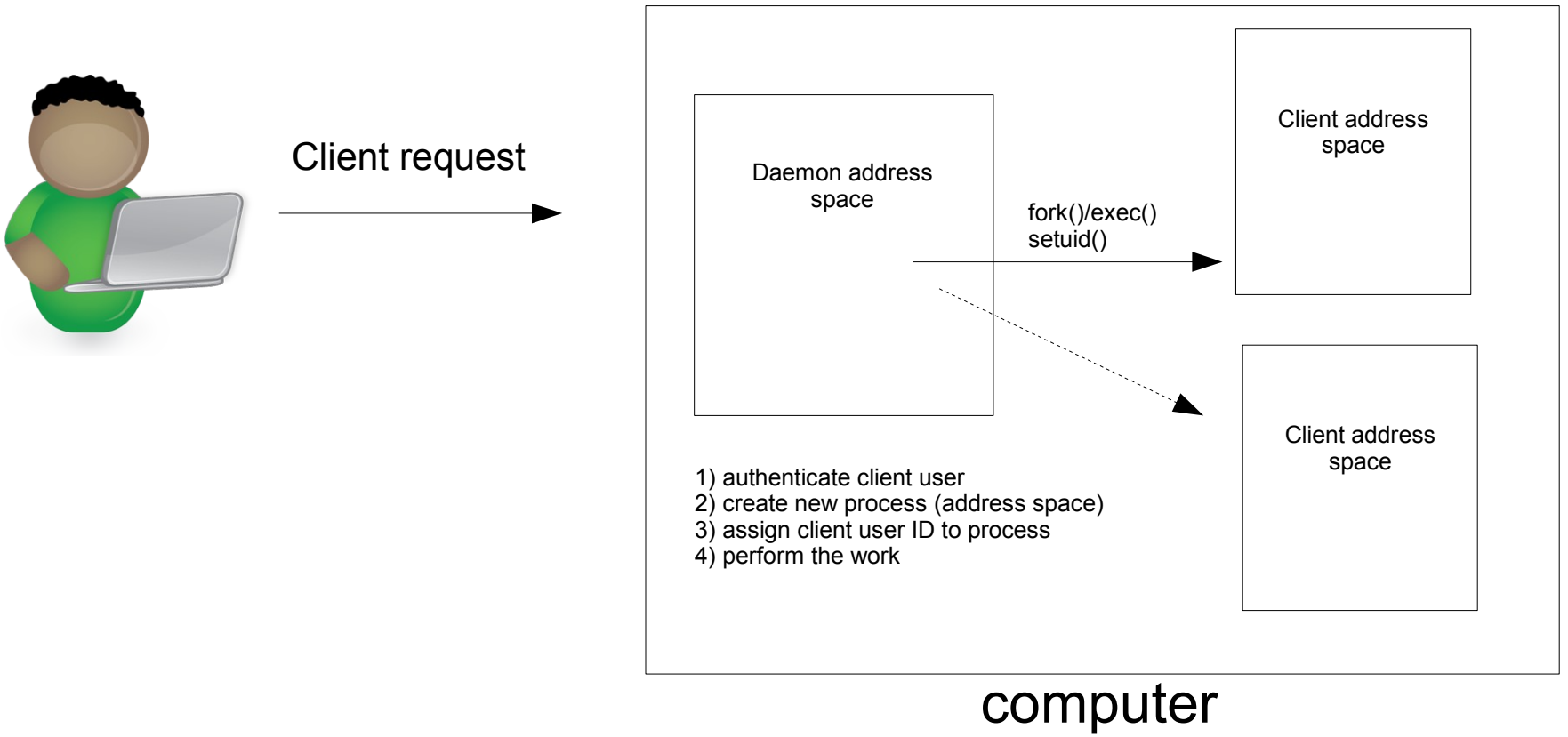
- UNIX services first introduced as OpenEdition on MVS/SP V4.3
- Adhered to the POSIX standard, calling for:
  - A hierarchical file system
  - A set of APIs
  - An interactive shell environment with a defined minimum set of commands and utilities

Background ...

- Required an interesting set of design and implementation decisions when merging the UNIX and MVS models
  - Does an API like setuid() simply change UID? Or should it also build an entire security environment (ACEE) for the target MVS user ID?
  - Do we provide various task-level services which don't exist on UNIX, but are heavily used on MVS?
  - Do we require APF authorization for identity-switching services?
  - Do we fully honor superuser or do we require additional/alternative authorization mechanisms?

Daemons...so what's a daemon?

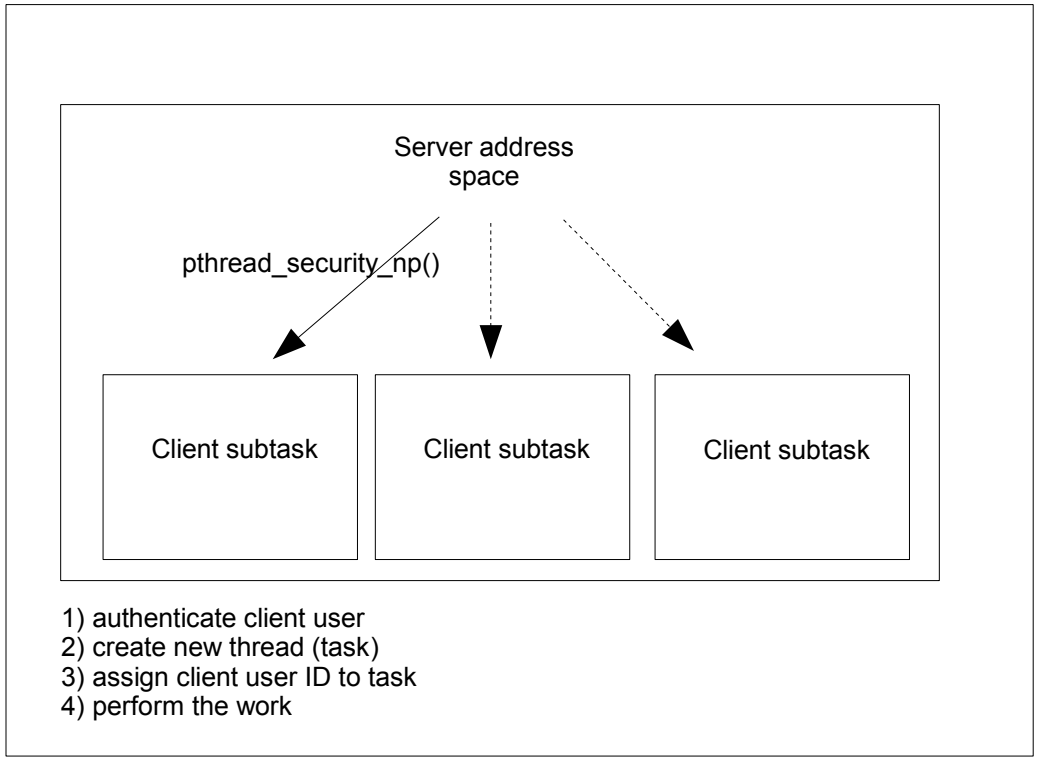
- This is a pretty typical UNIX programming model
- Long-running process that performs work on behalf of clients



Servers...so what's a server?

- This is a pretty typical MVS programming model. Not UNIX at all.
- Long-running process that performs work on behalf of clients!

Client request



computer

How are they identified?

- There is no 'server' or 'daemon' attribute that you assign to a given process (started task)
- It is more a function of the underlying services that they invoke
- Each service has its authorization rules
  - Which may require authorization to BPX.DAEMON or BPX.SERVER in the FACILTY class
  - Services documented in USS Assembler Callable Services Reference and C/C++ Run-Time Library Reference
- It's possible that a given application could invoke both types of services, and require both types of access
- Ultimately, you hope that the software you purchase correctly documents its requirements to your security administrators
- And if you are the one writing the application, you document your requirements based on the IBM documentation for the services you invoke

Daemons...examples

- Daemons supplied with z/OS UNIX:
  - inetd - the internet daemon
  - rlogind - the remote login daemon
  - cron - the batch scheduler
  - uucpd - the UUCP (UNIX-to-UNIX Copy Program) daemon
- Daemons supplied with IBM Communications Server for z/OS
  - syslog - message routing daemon
  - ftpd – file transfer protocol daemon
  - rshd - remote shell daemon
  - telnetd – telnet daemon
  - etc

How does a daemon change identity?

- Daemon programs call identity changing services to alter the UID and RACF user ID of an address space
  - seteuid()
  - setuid()
  - spawn() with a user ID
- Daemons can become any user that has an OMVS segment defined
  - any user if you have BPX.DEFAULT.USER (going away in V2R1) or BPX.UNIQUE.USER set up
    - You can block a user from being the target of an identity switch by assigning them an 'empty' OMVS segment

      ALTUSER JOE OMVS(NOUID)
    - But then they can't use UNIX at all

Identity switches: authenticated vs. unauthenticated

- Authenticated: prior to issuing the setuid() or pthread_security_np(), one of the following has occurred
  - The client password has been validated.
    - For example, by using the __passwd() function
  - The client has been authenticated via digital certificate
    - Using the __certificate() function
  - The daemon has SURROGAT authority to the user
    - READ access to BPX.SRV.userid

- Unauthenticated: none of these have occurred
  - Generally should only be used for anonymous type access to public information

Identity switches: authenticated vs. unauthenticated ...

- The BPX.SERVER and BPX.DAEMON controls are intended to add additional safeguards around **UN**authenticated identity switches
  - If an application (or user for that matter) has SURROGAT authority to another user, or knows that user's password, then it is trusted to act on behalf of that user

- BPX.DAEMON and BPX.SERVER provide additional controls over UID(0) humans, to prevent them from simply becoming another user ID
  - But as always, this is not 100% bullet-proof due to applications like cron which perform their work based on the contents of UNIX files, which a UID(0) person would be able to edit.

Daemons

- Three (two and a half?) levels of daemon security documented by IBM (there used to be two)
  - UNIX level
  - z/OS UNIX level
    - With BPX.DAEMON defined
    - With BPX.DAEMON, BPX.MAINCHECK and enhanced program security in effect

- Daemons do not need APF authorization regardless of the level

UNIX level

- UID(0) is sufficient
  - Comparable to traditional UNIX
  - Daemon can do anything, including identity switch, whether the target user has been authenticated or not

- This is the default. It is less secure.

- But if you are assigning UID(0) to humans (e.g. system programmers), then you have already granted the keys to your kingdom to those whom you probably trust less than the software you purchase

z/OS UNIX level

- Define BPX.DAEMON in the FACILITY class

- Daemon requires UID(0) **AND** READ access to BPX.DAEMON
  - Can still do unauthenticated identity switches, but a human not permitted to BPX.DAEMON cannot

- Daemon requires clean address space in order to invoke certain identity-related APIs
  - Requires all programs used by daemon to be identified to RACF program control
    - Programs in MVS libraries need to be covered by a PROGRAM class profile
    - Programs in the file system need to have the +p extended attribute

z/OS UNIX level with enhanced program security in effect and BPX.MAINCHECK defined

- Enhanced program security is used to identify programs that you expect users to execute
  - As opposed to “internal” programs called by such programs
  - Preventing the execution of “internal” programs directly by users protects against unpredictable side-effects which might ensue
  - Requires UNIX executables that use execute-controlled programs to be moved into an MVS library and identified as a MAIN program

- Defining BPX.MAINCHECK in the FACILITY class further extends this protection to UNIX programs when BPX.DAEMON is defined
  - Requires daemon's first (jobstep) program to be identified as MAIN, regardless of whether it calls execute-controlled programs

Additional consideration for daemons: BPXROOT

- A function like seteuid() changes UNIX identity (effective UID) to the value specified, but does not necessarily have a corresponding MVS user ID
- When the target is UID(0), there are many possible user IDs to choose from, each with the potential for radically different MVS resource access
  - Including to BPX.DAEMON itself!
- USS disambiguates this with a BPXROOT setting in BPXPRMxx
- The specified user ID becomes the target of such a switch, thus allowing you to scope and quantify the authority of the target user
- IBM tells you **not** to permit this user ID to BPX.DAEMON, so that the ability to further switch identities is prevented

Servers

- Three (two and a half?) levels of server security documented by IBM
  - UNIX level
  - z/OS UNIX level
    - With BPX.SERVER defined
    - With BPX.SERVER, BPX.MAINCHECK and enhanced program security in effect

- Servers do not need APF authorization regardless of the level

UNIX level

- UID(0) is sufficient
  - Server can do anything, including running work under another identity, whether the target user has been authenticated or not

- This is the default. It is less secure.

- But if you are assigning UID(0) to humans (e.g. system programmers), then you have already granted the keys to your kingdom to those whom you probably trust less than the software you purchase

z/OS UNIX level

- Define BPX.SERVER in the FACILITY class

- Server requires access to BPX.SERVER ***but not UID(0)***
  - READ: If client not authenticated, then both the client and server identity require access to resources
  - UPDATE: Only client requires access to resources, regardless of whether it is authenticated

- Server requires clean address space

z/OS UNIX level with enhanced program security in effect and BPX.MAINCHECK defined

- As above for daemons

Additional consideration for servers: resource limits

- With multiple threads running in an address space, the default per-user resource limits defined in BPXPRMxx (PARMLIB) could be exceeded
- Increased limits can be specified in the server identity's OMVS segment, so that the server does not need UID(0) in order to exceed the defaults
  - ASSIZEMAX(address-space-size)
  - CPUTIMEMAX(cpu-time)
  - FILEPROCMAX(files-per-process)
  - MEMLIMIT(nonshared-memory-size)
  - MMAPAREAMAX(memory-map-size)
  - PROCUSERMAX(processes-per-UID)
  - SHMEMMAX(shared memory size)
  - THREADSMAX(threads-per-process)

References

- “Setting up for daemons” and “Preparing security for servers” chapters in z/OS UNIX System Services Planning
- z/OS UNIX System Services Programming: Assembler Callable Services Reference
- C/C++ Run-Time Library Reference
- “Protecting programs” chapter in RACF Security Administrator's Guide
- The RedBook titled Sysplex eBusiness Security z/OS V1R7 Update has some information in Chapter 3: UNIX System Services Security

API Reference

(hover over service names for hyperlink)

C/C++ api	Unix assembler service	description
pthread_security_np	BPX1TLS	Create or delete thread-level security
__certificate	BPX1SEC	De/register or authenticate a certificate
__must_stay_clean	BPX1ENV	Query or enable the “must stay clean” state for a process
__login	BPX1SEC	Provide a password or BPX.DAEMON or UID(0) If BPX.DAEMON, then clean addr space
__passwd	BPX1PWD	Verify or change user password
seteuid	BPX1SEU	Set the effective UID
setreuid	BPX1SRU	Set real and effective UIDs
setuid	BPX1SUI	Set UIDs
__check_resource_ath_np	BPX1ACK (removed in R13)	Determine access to MVS resources
connect		Connect a socket