

Started on	Tuesday, 16 January 2024, 5:08 PM
State	Finished
Completed on	Tuesday, 16 January 2024, 5:52 PM
Time taken	44 mins 52 secs
Grade	10.20 out of 15.00 (68%)

Question **1**
 Incorrect
 Mark 0.00 out of 1.00

Select all the correct statements about the process init on Linuxes/Unixes.

Select one or more:

- ☒
 a. init can not be killed with SIGKILL✔
- ☐
 b. only a process run by 'root' user can exec 'init'
- ☐
 c. init is created by kernel 'by hand'
- ☒
 d. any user process can fork and exec init✖
- ☒
 e. init typically has a pid=1✔
- ☒
 f. init is created by kernel by forking itself✖
- ☐
 g. no process can exec 'init'

Your answer is incorrect.

The correct answers are: init is created by kernel 'by hand', init typically has a pid=1, init can not be killed with SIGKILL, only a process run by 'root' user can exec 'init'

Question **2**
 Incorrect
 Mark 0.00 out of 1.00

Write the possible contents of the file /tmp/xyz after this program.

In the answer if you want to mention any non-text character, then write \0 For example abcl0l0 means abc followed by any two non-text characters

```

int main(int argc, char *argv[]) {
    int fd1, fd2, n, i;
    char buf[128];

    fd1 = open("/tmp/xyz", O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);
    write(fd1, "hello", 5);
    fd2 = open("/tmp/xyz", O_WRONLY, S_IRUSR|S_IWUSR);
    write(fd2, "bye", 3);
    close(fd1);
    close(fd2);
    return 0;
}
    
```

Answer:
✖

The correct answer is: byelo

Question **3**

Partially correct

Mark 0.60 out of 1.00

Select all the correct statements about two modes of CPU operation

Select one or more:

- ☐ a. Some instructions are allowed to run only in user mode, while all instructions can run in kernel mode
- ☒ b. There is an instruction like 'iret' to return from kernel mode to user mode ✓
- ☒ c. The two modes are essential for a multitasking system ✓
- ☒ d. The two modes are essential for a multiprogramming system ✓
- ☐ e. The software interrupt instructions change the mode from user mode to kernel mode and jumps to predefined location simultaneously

Your answer is partially correct.

You have correctly selected 3.

The correct answers are: The two modes are essential for a multiprogramming system, The two modes are essential for a multitasking system, There is an instruction like 'iret' to return from kernel mode to user mode, The software interrupt instructions change the mode from user mode to kernel mode and jumps to predefined location simultaneously, Some instructions are allowed to run only in user mode, while all instructions can run in kernel mode

Question **4**

Correct

Mark 0.50 out of 0.50

Compare multiprogramming with multitasking

- ☐ a. A multitasking system is not necessarily multiprogramming
- ☒ b. A multiprogramming system is not necessarily multitasking ✓

The correct answer is: A multiprogramming system is not necessarily multitasking

Question 5

Partially correct

Mark 0.25 out of 1.00

Given below is the output of "ps -eaf".

Answer the questions based on it.

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	Jan05	?	00:01:08	/sbin/init splash
root	2	0	0	Jan05	?	00:00:00	[kthreadd]
root	3	2	0	Jan05	?	00:00:00	[rcu_gp]
root	4	2	0	Jan05	?	00:00:00	[rcu_par_gp]
root	9	2	0	Jan05	?	00:00:00	[mm_percpu_wq]
root	10	2	0	Jan05	?	00:00:00	[rcu_tasks_rude_]
root	11	2	0	Jan05	?	00:00:00	[rcu_tasks_trace]
root	12	2	0	Jan05	?	00:00:22	[ksoftirqd/0]
root	13	2	0	Jan05	?	00:06:29	[rcu_sched]
root	14	2	0	Jan05	?	00:00:02	[migration/0]
root	15	2	0	Jan05	?	00:00:00	[idle_inject/0]
root	16	2	0	Jan05	?	00:00:00	[cpuhp/0]
root	17	2	0	Jan05	?	00:00:00	[cpuhp/1]
root	18	2	0	Jan05	?	00:00:00	[idle_inject/1]
root	19	2	0	Jan05	?	00:00:03	[migration/1]
root	20	2	0	Jan05	?	00:00:13	[ksoftirqd/1]
root	22	2	0	Jan05	?	00:00:00	[kworker/1:0H-events_highpri]
root	23	2	0	Jan05	?	00:00:00	[cpuhp/2]
root	24	2	0	Jan05	?	00:00:00	[idle_inject/2]
root	25	2	0	Jan05	?	00:00:01	[migration/2]
root	26	2	0	Jan05	?	00:00:09	[ksoftirqd/2]
root	28	2	0	Jan05	?	00:00:00	[kworker/2:0H-kblockd]
root	29	2	0	Jan05	?	00:00:00	[cpuhp/3]
root	30	2	0	Jan05	?	00:00:00	[idle_inject/3]
root	31	2	0	Jan05	?	00:00:02	[migration/3]
root	32	2	0	Jan05	?	00:00:07	[ksoftirqd/3]
root	34	2	0	Jan05	?	00:00:00	[kworker/3:0H-events_highpri]
root	35	2	0	Jan05	?	00:00:00	[cpuhp/4]
root	36	2	0	Jan05	?	00:00:00	[idle_inject/4]
root	37	2	0	Jan05	?	00:00:02	[migration/4]
root	38	2	0	Jan05	?	00:00:06	[ksoftirqd/4]
root	40	2	0	Jan05	?	00:00:00	[kworker/4:0H-events_highpri]
root	41	2	0	Jan05	?	00:00:00	[cpuhp/5]
root	42	2	0	Jan05	?	00:00:00	[idle_inject/5]
root	43	2	0	Jan05	?	00:00:02	[migration/5]
root	44	2	0	Jan05	?	00:00:05	[ksoftirqd/5]
root	46	2	0	Jan05	?	00:00:00	[kworker/5:0H-events_highpri]
root	47	2	0	Jan05	?	00:00:00	[cpuhp/6]
root	48	2	0	Jan05	?	00:00:00	[idle_inject/6]
root	49	2	0	Jan05	?	00:00:02	[migration/6]
root	50	2	0	Jan05	?	00:00:05	[ksoftirqd/6]
root	52	2	0	Jan05	?	00:00:00	[kworker/6:0H-events_highpri]
root	53	2	0	Jan05	?	00:00:00	[cpuhp/7]
root	54	2	0	Jan05	?	00:00:00	[idle_inject/7]
root	55	2	0	Jan05	?	00:00:02	[migration/7]
root	56	2	0	Jan05	?	00:00:06	[ksoftirqd/7]
root	58	2	0	Jan05	?	00:00:00	[kworker/7:0H-events_highpri]
root	59	2	0	Jan05	?	00:00:00	[kdevtmpfs]
root	60	2	0	Jan05	?	00:00:00	[netns]
root	61	2	0	Jan05	?	00:00:00	[inet_frag_wq]
root	62	2	0	Jan05	?	00:00:00	[kauditd]
root	63	2	0	Jan05	?	00:00:00	[khungtaskd]
root	64	2	0	Jan05	?	00:00:00	[oom_reaper]
root	65	2	0	Jan05	?	00:00:00	[writeback]
root	66	2	0	Jan05	?	00:01:58	[kcompactd0]
root	67	2	0	Jan05	?	00:00:00	[ksmd]
root	68	2	0	Jan05	?	00:00:04	[khugepaged]
root	115	2	0	Jan05	?	00:00:00	[kintegrityd]
root	116	2	0	Jan05	?	00:00:00	[kblockd]
root	117	2	0	Jan05	?	00:00:00	[blkcg_punt_bio]
root	118	2	0	Jan05	?	00:00:00	[tpm_dev_wq]

root	119	2	0	Jan05	?	00:00:00	[ata_sff]
root	120	2	0	Jan05	?	00:00:00	[md]
root	121	2	0	Jan05	?	00:00:00	[edac-poller]
root	122	2	0	Jan05	?	00:00:00	[devfreq_wq]
root	123	2	0	Jan05	?	00:00:00	[watchdogd]
root	129	2	0	Jan05	?	00:00:00	[irq/25-AMD-Vi]
root	131	2	0	Jan05	?	00:04:33	[kswapd0]
root	132	2	0	Jan05	?	00:00:00	[ecryptfs-kthrea]
root	134	2	0	Jan05	?	00:00:00	[kthrotld]
root	135	2	0	Jan05	?	00:00:00	[irq/27-pciehp]
root	139	2	0	Jan05	?	00:00:00	[acpi_thermal_pm]
root	140	2	0	Jan05	?	00:00:00	[vfio-irqfd-clea]
root	141	2	0	Jan05	?	00:00:00	[ipv6_addrconf]
root	144	2	0	Jan05	?	00:00:03	[kworker/6:1H-kblockd]
root	151	2	0	Jan05	?	00:00:00	[kstrp]
root	154	2	0	Jan05	?	00:00:00	[zswap-shrink]
root	162	2	0	Jan05	?	00:00:00	[charger_manager]
root	164	2	0	Jan05	?	00:00:03	[kworker/4:1H-kblockd]
root	197	2	0	Jan05	?	00:00:03	[kworker/3:1H-kblockd]
root	213	2	0	Jan05	?	00:00:03	[kworker/7:1H-kblockd]
root	215	2	0	Jan05	?	00:00:00	[nvme-wq]
root	216	2	0	Jan05	?	00:00:00	[nvme-reset-wq]
root	217	2	0	Jan05	?	00:00:00	[nvme-delete-wq]
root	223	2	0	Jan05	?	00:00:00	[irq/42-ELAN2513]
root	224	2	0	Jan05	?	00:04:20	[irq/41-ELAN071B]
root	225	2	0	Jan05	?	00:00:00	[cryptd]
root	226	2	0	Jan05	?	00:00:00	[amd_iommu_v2]
root	249	2	0	Jan05	?	00:00:00	[ttm_swap]
root	250	2	0	Jan05	?	00:20:29	[gfx]
root	251	2	0	Jan05	?	00:00:00	[comp_1.0.0]
root	252	2	0	Jan05	?	00:00:00	[comp_1.1.0]
root	253	2	0	Jan05	?	00:00:00	[comp_1.2.0]
root	254	2	0	Jan05	?	00:00:00	[comp_1.3.0]
root	255	2	0	Jan05	?	00:00:00	[comp_1.0.1]
root	256	2	0	Jan05	?	00:00:00	[comp_1.1.1]
root	257	2	0	Jan05	?	00:00:00	[comp_1.2.1]
root	258	2	0	Jan05	?	00:00:00	[comp_1.3.1]
root	259	2	0	Jan05	?	00:00:27	[sdma0]
root	260	2	0	Jan05	?	00:00:00	[vcn_dec]
root	261	2	0	Jan05	?	00:00:00	[vcn_enc0]
root	262	2	0	Jan05	?	00:00:00	[vcn_enc1]
root	263	2	0	Jan05	?	00:00:00	[jpeg_dec]
root	265	2	0	Jan05	?	00:00:00	[card0-crtc0]
root	266	2	0	Jan05	?	00:00:00	[card0-crtc1]
root	267	2	0	Jan05	?	00:00:00	[card0-crtc2]
root	268	2	0	Jan05	?	00:00:00	[card0-crtc3]
root	271	2	0	Jan05	?	00:00:03	[kworker/5:1H-kblockd]
root	277	2	0	Jan05	?	00:00:03	[kworker/1:1H-kblockd]
root	320	2	0	Jan05	?	00:00:00	[raid5wq]
root	380	2	0	Jan05	?	00:00:11	[jbd2/nvme0n1p5-]
root	381	2	0	Jan05	?	00:00:00	[ext4-rsv-conver]
root	432	2	0	Jan05	?	00:00:03	[kworker/2:1H-kblockd]
root	446	1	0	Jan05	?	00:01:16	/lib/systemd/systemd-journald
root	470	2	0	Jan05	?	00:00:00	[rpciod]
root	473	2	0	Jan05	?	00:00:00	[xprtiod]
root	474	2	0	Jan05	?	00:00:00	bpfilter_umh
root	503	1	0	Jan05	?	00:00:07	/lib/systemd/systemd-udev
root	517	2	0	Jan05	?	00:00:00	[loop0]
root	554	2	0	Jan05	?	00:00:00	[loop1]
root	563	2	0	Jan05	?	00:00:00	[loop2]
root	586	2	0	Jan05	?	00:00:00	[loop3]
root	588	2	0	Jan05	?	00:00:00	[loop4]
root	589	2	0	Jan05	?	00:00:00	[loop5]
root	612	2	0	Jan05	?	00:00:00	[loop6]
root	613	2	0	Jan05	?	00:00:00	[loop7]
root	629	2	0	Jan05	?	00:00:00	[loop8]
root	637	2	0	Jan05	?	00:00:00	[cfg80211]
root	676	2	0	Jan05	?	00:05:00	[irq/75-iwlwifi:]
root	678	2	0	Jan05	?	00:01:07	[irq/76-iwlwifi:]
root	682	2	0	Jan05	?	00:01:27	[irq/77-iwlwifi:]

```

root      688      2 0 Jan05 ?      00:00:49 [irq/78-iwlwifi:]
root      695      2 0 Jan05 ?      00:01:39 [irq/79-iwlwifi:]
root      700      2 0 Jan05 ?      00:01:22 [irq/80-iwlwifi:]
root      703      2 0 Jan05 ?      00:01:13 [irq/81-iwlwifi:]
root      704      2 0 Jan05 ?      00:01:38 [irq/82-iwlwifi:]
root      708      2 0 Jan05 ?      00:00:44 [irq/83-iwlwifi:]
root      713      2 0 Jan05 ?      00:00:00 [loop9]
root      715      2 0 Jan05 ?      00:00:00 [irq/84-iwlwifi:]
root      782      2 0 Jan05 ?      00:00:00 [loop10]
root      797      2 0 Jan05 ?      00:00:00 [loop11]
root      811      2 0 Jan05 ?      00:00:00 [loop12]
root      838      2 0 Jan05 ?      00:00:00 [loop13]
root      847      2 0 Jan05 ?      00:00:00 [loop14]
root      879      2 0 Jan05 ?      00:00:00 [loop15]
root      884      2 0 Jan05 ?      00:00:00 [loop16]
root      885      2 0 Jan05 ?      00:00:00 [loop17]
root      945      2 0 Jan05 ?      00:00:00 [loop18]
root      946      2 0 Jan05 ?      00:00:00 [loop19]
root      947      2 0 Jan05 ?      00:00:00 [loop20]
root     1012      2 0 Jan05 ?      00:00:00 [jbd2/nvme0n1p8-]
root     1013      2 0 Jan05 ?      00:00:00 [ext4-rsv-conver]
root     1015      2 0 Jan05 ?      00:01:09 [jbd2/nvme0n1p7-]
root     1016      2 0 Jan05 ?      00:00:00 [ext4-rsv-conver]
_rpc     1062      1 0 Jan05 ?      00:00:00 /sbin/rpcbind -f -w
systemd+ 1063      1 0 Jan05 ?      00:01:24 /lib/systemd/systemd-resolved
systemd+ 1064      1 0 Jan05 ?      00:00:00 /lib/systemd/systemd-timesyncd
root     1144      1 0 Jan05 ?      00:00:46 /usr/sbin/acpid
avahi    1146      1 0 Jan05 ?      00:00:06 avahi-daemon: running [abhijit-laptop.local]
root     1149      1 0 Jan05 ?      00:00:01 /usr/lib/bluetooth/bluetoothd
message+ 1150      1 0 Jan05 ?      00:04:21 /usr/bin/dbus-daemon --system --address=systemd: --nofork --
nopicfile --systemd-activation --syslog-only
root     1152      1 0 Jan05 ?      00:03:12 /usr/sbin/NetworkManager --no-daemon
root     1157      1 0 Jan05 ?      00:01:02 /usr/sbin/iio-sensor-proxy
root     1159      1 0 Jan05 ?      00:00:27 /usr/sbin/irqbalance --foreground
root     1162      1 0 Jan05 ?      00:00:01 /usr/bin/lxcfs /var/lib/lxcfs
root     1165      1 0 Jan05 ?      00:00:00 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-
triggers
root     1170      1 0 Jan05 ?      00:00:29 /usr/lib/policykit-1/polkitd --no-debug
syslog   1175      1 0 Jan05 ?      00:00:20 /usr/sbin/rsyslogd -n -iNONE
root     1182      1 0 Jan05 ?      00:01:13 /usr/lib/snapd/snapd
root     1187      1 0 Jan05 ?      00:00:12 /usr/lib/accounts-service/accounts-daemon
root     1192      1 0 Jan05 ?      00:00:00 /usr/sbin/cron -f
root     1198      1 0 Jan05 ?      00:00:00 /usr/libexec/switcheroo-control
root     1201      1 0 Jan05 ?      00:00:12 /lib/systemd/systemd-logind
root     1202      1 0 Jan05 ?      00:00:07 /lib/systemd/systemd-machined
root     1203      1 0 Jan05 ?      00:01:28 /usr/lib/udisks2/udisksd
root     1204      1 0 Jan05 ?      00:00:15 /sbin/wpa_supplicant -u -s -O /run/wpa_supplicant
daemon   1209      1 0 Jan05 ?      00:00:00 /usr/sbin/atd -f
avahi    1216      1146 0 Jan05 ?      00:00:00 avahi-daemon: chroot helper
docker-+ 1279      1 0 Jan05 ?      00:00:22 /usr/bin/docker-registry serve /etc/docker/registry/config.yml
root     1282      1 0 Jan05 ?      00:00:00 /usr/bin/python3 /usr/bin/twisted3 --nodaemon --pidfile= epoptes
jenkins   1285      1 0 Jan05 ?      00:15:01 /usr/bin/java -Djava.awt.headless=true -jar
/usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080
root     1296      1 0 Jan05 ?      00:00:18 php-fpm: master process (/etc/php/7.4/fpm/php-fpm.conf)
vnstat   1314      1 0 Jan05 ?      00:00:15 /usr/sbin/vnstatd -n
root     1326      1 0 Jan05 ?      00:00:02 /usr/sbin/ModemManager
root     1327      1 0 Jan05 ?      00:00:31 /usr/bin/anydesk --service
colord   1359      1 0 Jan05 ?      00:00:01 /usr/libexec/colord
root     1374      1 0 Jan05 ?      00:00:00 /usr/sbin/gdm3
root     1420      1 0 Jan05 ?      00:00:14 /usr/sbin/apache2 -k start
www-data 1436      1296 0 Jan05 ?      00:00:00 php-fpm: pool www
www-data 1437      1296 0 Jan05 ?      00:00:00 php-fpm: pool www
mysql    1461      1 0 Jan05 ?      00:38:52 /usr/sbin/mysqld
root     1490      1 0 Jan05 ?      00:00:04 /usr/sbin/libvirtd
root     1491      1 0 Jan05 ?      00:00:00 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-
shutdown --wait-for-signal
rtkit    1593      1 0 Jan05 ?      00:00:07 /usr/libexec/rtkit-daemon
libvirt+ 1766      1 0 Jan05 ?      00:00:00 /usr/sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/default.conf
--leasefile-ro --dhcp-script=/usr/lib/libvirt/libvirt_leaseshelper
root     1767      1766 0 Jan05 ?      00:00:00 /usr/sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/default.conf

```

```

--leasefile-ro --dhcp-script=/usr/lib/libvirt/libvirt_leaseshelper
root      1859      1 0 Jan05 ?      00:00:18 /usr/lib/upower/upowerd
root      1995      1 0 Jan05 ?      00:00:00 /opt/saltstack/salt/run/run minion
root      2041     1995 0 Jan05 ?      00:05:18 /opt/saltstack/salt/run/run minion MultiMinionProcessManager
MinionProcessManager
root      2278      1 0 Jan05 ?      00:00:50 /usr/bin/dockerd -H fd:// --
containerd=/run/containerd/containerd.sock
root      2282      1 0 Jan05 ?      00:00:17 /usr/sbin/inetd
whoopsie  2302      1 0 Jan05 ?      00:00:01 /usr/bin/whoopsie -f
kernoops  2330      1 0 Jan05 ?      00:00:19 /usr/sbin/kerneloops --test
kernoops  2341      1 0 Jan05 ?      00:00:19 /usr/sbin/kerneloops
root      2366      2 0 Jan05 ?      00:00:00 [iprt-VBoxWQueue]
lxc-dns+  2370      1 0 Jan05 ?      00:00:00 dnsmasq --conf-file=/dev/null -u lxc-dnsmasq --strict-order --bind-
interfaces --pid-file=/run/lxc/dnsmasq.pid --listen-address 10.0.3.1 --dhcp-range 10.0.3.2,10.0.3.254 --dhcp-lease-
max=253 --dhcp-no-override --except-interface=lo --interface=lxcbr0 --dhcp-
leasefile=/var/lib/misc/dnsmasq.lxcbr0.leases --dhcp-authoritative
root      2404      2 0 Jan05 ?      00:00:00 [iprt-VBoxTscThr]
root      3508      1 0 Jan05 ?      00:00:01 /usr/lib/postfix/sbin/master -w
root      3615     1374 0 Jan05 ?      00:00:01 gdm-session-worker [pam/gdm-password]
abhijit   3629      1 0 Jan05 ?      00:00:17 /lib/systemd/systemd --user
abhijit   3630     3629 0 Jan05 ?      00:00:00 (sd-pam)
abhijit   3636     3629 1 Jan05 ?      04:31:21 /usr/bin/pulseaudio --daemonize=no --log-target=journal
abhijit   3638     3629 0 Jan05 ?      00:20:35 /usr/libexec/tracker-miner-fs
abhijit   3642     3629 0 Jan05 ?      00:01:11 /usr/bin/dbus-daemon --session --address=systemd: --nofork --
nopicfile --systemd-activation --syslog-only
abhijit   3644      1 0 Jan05 ?      00:00:03 /usr/bin/gnome-keyring-daemon --daemonize --login
abhijit   3662     3629 0 Jan05 ?      00:00:01 /usr/libexec/gvfsd
abhijit   3667     3629 0 Jan05 ?      00:00:00 /usr/libexec/gvfsd-fuse /run/user/1000/gvfs -f -o big_writes
abhijit   3673     3629 0 Jan05 ?      00:00:02 /usr/libexec/gvfs-udisks2-volume-monitor
abhijit   3681     3629 0 Jan05 ?      00:00:01 /usr/libexec/gvfs-mtp-volume-monitor
abhijit   3685     3629 0 Jan05 ?      00:00:11 /usr/libexec/gvfs-afc-volume-monitor
abhijit   3690     3629 0 Jan05 ?      00:00:01 /usr/libexec/gvfs-gphoto2-volume-monitor
abhijit   3695     3629 0 Jan05 ?      00:00:01 /usr/libexec/gvfs-goa-volume-monitor
abhijit   3700     3629 0 Jan05 ?      00:00:01 /usr/libexec/goa-daemon
root      3701      2 0 Jan05 ?      00:00:00 [krfcommd]
abhijit   3708     3629 0 Jan05 ?      00:00:04 /usr/libexec/goa-identity-service
abhijit   3724     3615 0 Jan05 tty2    00:00:00 /usr/lib/gdm3/gdm-x-session --run-script env
GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu
abhijit   3726     3724 1 Jan05 tty2    02:47:57 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth
/run/user/1000/gdm/Xauthority -background none -noreset -keeptty -verbose 3
abhijit   3747     3724 0 Jan05 tty2    00:00:00 /usr/libexec/gnome-session-binary --systemd --systemd --
session=ubuntu
abhijit   3816     3747 0 Jan05 ?      00:00:01 /usr/bin/ssh-agent /usr/bin/im-launch env
GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu
abhijit   3845     3629 0 Jan05 ?      00:00:00 /usr/libexec/at-spi-bus-launcher
abhijit   3850     3845 0 Jan05 ?      00:00:07 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-
spi2/accessibility.conf --nofork --print-address 3
abhijit   3869     3629 0 Jan05 ?      00:00:00 /usr/libexec/gnome-session-ctl --monitor
abhijit   3876     3629 0 Jan05 ?      00:00:04 /usr/libexec/gnome-session-binary --systemd-service --
session=ubuntu
abhijit   3890     3629 1 Jan05 ?      03:43:47 /usr/bin/gnome-shell
abhijit   3927     3890 0 Jan05 ?      00:31:49 ibus-daemon --panel disable --xim
abhijit   3931     3927 0 Jan05 ?      00:00:00 /usr/libexec/ibus-dconf
abhijit   3932     3927 0 Jan05 ?      00:01:51 /usr/libexec/ibus-extension-gtk3
abhijit   3934     3629 0 Jan05 ?      00:00:04 /usr/libexec/ibus-x11 --kill-daemon
abhijit   3937     3629 0 Jan05 ?      00:00:02 /usr/libexec/ibus-portal
abhijit   3949     3629 0 Jan05 ?      00:00:27 /usr/libexec/at-spi2-registryd --use-gnome-session
abhijit   3953     3629 0 Jan05 ?      00:00:00 /usr/libexec/xdg-permission-store
abhijit   3955     3629 0 Jan05 ?      00:00:01 /usr/libexec/gnome-shell-calendar-server
abhijit   3964     3629 0 Jan05 ?      00:00:00 /usr/libexec/evolution-source-registry
abhijit   3973     3629 0 Jan05 ?      00:00:02 /usr/libexec/evolution-calendar-factory
abhijit   3986     3629 0 Jan05 ?      00:00:01 /usr/libexec/dconf-service
abhijit   3992     3629 0 Jan05 ?      00:00:01 /usr/libexec/evolution-addressbook-factory
abhijit   4007     3629 0 Jan05 ?      00:00:00 /usr/bin/gjs /usr/share/gnome-shell/org.gnome.Shell.Notifications
abhijit   4023     3629 0 Jan05 ?      00:00:00 /usr/libexec/gsd-a11y-settings
abhijit   4025     3629 0 Jan05 ?      00:00:08 /usr/libexec/gsd-color
abhijit   4029     3629 0 Jan05 ?      00:00:00 /usr/libexec/gsd-datetime
abhijit   4032     3629 0 Jan05 ?      00:00:21 /usr/libexec/gsd-housekeeping
abhijit   4033     3629 0 Jan05 ?      00:00:05 /usr/libexec/gsd-keyboard
abhijit   4036     3629 0 Jan05 ?      00:00:12 /usr/libexec/gsd-media-keys

```

abhijit	4037	3629	0	Jan05	?	00:00:11	/usr/libexec/gsd-power
abhijit	4038	3629	0	Jan05	?	00:00:00	/usr/libexec/gsd-print-notifications
abhijit	4039	3629	0	Jan05	?	00:00:01	/usr/libexec/gsd-rfkill
abhijit	4041	3629	0	Jan05	?	00:00:01	/usr/libexec/gsd-screensaver-proxy
abhijit	4042	3876	0	Jan05	?	00:01:06	/usr/lib/x86_64-linux-gnu/libexec/kdeconnectd
abhijit	4045	3629	0	Jan05	?	00:00:35	/usr/libexec/gsd-sharing
abhijit	4047	3629	0	Jan05	?	00:00:00	/usr/libexec/gsd-smartcard
abhijit	4051	3629	0	Jan05	?	00:00:00	/usr/libexec/gsd-sound
abhijit	4057	3629	0	Jan05	?	00:00:00	/usr/libexec/gsd-usb-protection
abhijit	4063	3629	0	Jan05	?	00:00:05	/usr/libexec/gsd-wacom
abhijit	4071	3629	0	Jan05	?	00:00:00	/usr/libexec/gsd-wwan
abhijit	4072	3876	0	Jan05	?	00:01:00	baloo_file
abhijit	4075	3876	0	Jan05	?	00:00:00	/usr/libexec/gsd-disk-utility-notify
abhijit	4076	3629	0	Jan05	?	00:00:08	/usr/libexec/gsd-xsettings
abhijit	4078	3876	0	Jan05	?	00:00:10	/usr/bin/python3 /usr/bin/blueman-applet
abhijit	4082	3876	0	Jan05	?	00:00:14	/usr/bin/anydesk --tray
abhijit	4108	3876	0	Jan05	?	00:00:00	/usr/lib/x86_64-linux-gnu/indicator-messages/indicator-messages-service
abhijit	4109	3876	0	Jan05	?	00:00:06	/usr/libexec/evolution-data-server/evolution-alarm-notify
abhijit	4129	3629	0	Jan05	?	00:00:50	/snap/snap-store/959/usr/bin/snap-store --gaplication-service
abhijit	4191	3629	0	Jan05	?	00:00:00	/usr/libexec/gsd-printer
abhijit	4219	3629	0	Jan05	?	00:00:03	/usr/libexec/xdg-document-portal
abhijit	4265	3927	0	Jan05	?	00:03:20	/usr/libexec/ibus-engine-simple
abhijit	4291	3629	0	Jan05	?	00:00:10	/usr/bin/python3 /usr/bin/blueman-tray
abhijit	4301	3629	0	Jan05	?	00:00:00	/usr/lib/bluetooth/obexd
abhijit	4377	3662	0	Jan05	?	00:00:02	/usr/libexec/gvfsd-trash --spawner :1.3 /org/gtk/gvfs/exec_spaw/0
abhijit	4395	3629	0	Jan05	?	00:00:12	/usr/libexec/xdg-desktop-portal
abhijit	4399	3629	0	Jan05	?	00:01:08	/usr/libexec/xdg-desktop-portal-gtk
abhijit	4480	3629	0	Jan05	?	00:00:21	/usr/libexec/gvfsd-metadata
abhijit	5687	3662	0	Jan05	?	00:00:00	/usr/libexec/gvfsd-network --spawner :1.3 /org/gtk/gvfs/exec_spaw/1
abhijit	5701	3662	0	Jan05	?	00:00:01	/usr/libexec/gvfsd-dnssd --spawner :1.3 /org/gtk/gvfs/exec_spaw/3
root	6228	2	0	Jan05	?	00:00:00	[kdmflush]
root	6236	2	0	Jan05	?	00:00:00	[kcryptd_io/253:]
root	6237	2	0	Jan05	?	00:00:00	[kcryptd/253:0]
root	6238	2	0	Jan05	?	00:00:23	[dmccrypt_write/2]
root	6260	2	0	Jan05	?	00:00:24	[jbd2/dm-0-8]
root	6261	2	0	Jan05	?	00:00:00	[ext4-rsv-conver]
abhijit	6421	3927	0	Jan05	?	00:00:36	/usr/lib/ibus/ibus-engine-m17n --ibus
abhijit	6434	3876	0	Jan05	?	00:00:13	update-notifier
abhijit	30565	3629	0	Jan05	?	00:08:56	/usr/libexec/gnome-terminal-server
abhijit	30576	30565	0	Jan05	pts/0	00:00:00	bash
abhijit	131017	364845	1	Jan09	?	03:12:27	/usr/lib/virtualbox/VirtualBoxVM --comment ubuntu 18.04 --startvm 45993a5c-3ded-452f-941e-4579d12c1ad9 --no-startvm-errormsgbox
abhijit	159668	30565	0	Jan09	pts/10	00:00:00	bash
abhijit	161637	30565	0	Jan05	pts/1	00:00:01	bash
abhijit	171109	159668	0	Jan09	pts/10	00:00:33	evince.....
abhijit	171114	3629	0	Jan09	?	00:00:00	/usr/libexec/evince
abhijit	204055	3629	0	Jan10	?	00:00:13	/usr/bin/python3 /usr/bin/update-manager --no-update --no-focus-on-map
abhijit	270190	3629	0	Jan05	?	00:56:34	/usr/lib/x86_64-linux-gnu/libexec/kactivitymanagerd
abhijit	270199	3629	0	Jan05	?	00:00:24	/usr/bin/kglobalaccel5
abhijit	270207	3629	0	Jan05	?	00:00:00	kdeinit5: Running...
abhijit	270208	270207	0	Jan05	?	00:00:19	/usr/lib/x86_64-linux-gnu/libexec/kf5/klauncher --fd=8
abhijit	279971	3629	0	Jan05	?	01:20:12	telegram-desktop
abhijit	280011	279971	0	Jan05	?	00:00:00	sh -c /usr/lib/x86_64-linux-gnu/libproxy/0.4.15/pxgsettings org.gnome.system.proxy org.gnome.system.proxy.http org.gnome.system.proxy.https org.gnome.system


```
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {4dbb54ee-e1a2-41e4-b10b-587526532b78} 351306
true tab
abhijit 351495 351306 0 Jan12 ? 00:04:06 /usr/lib/firefox/firefox -contentproc -childID 2 -isForBrowser -
prefsLen 38090 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {02f1a4e7-d831-4a5f-a9fd-59a809bb03e8} 351306
true tab
abhijit 351717 351306 0 Jan12 ? 00:01:14 /usr/lib/firefox/firefox -contentproc -parentBuildID 20240108143603
-sandboxingKind 0 -prefsLen 42823 -prefMapSize 247458 -appDir /usr/lib/firefox/browser {a1e46009-cfd1-46c8-ac9c-
19ba8bf3f46a} 351306 true utility
abhijit 351844 351306 0 Jan12 ? 00:07:23 /usr/lib/firefox/firefox -contentproc -parentBuildID 20240108143603
-prefsLen 43306 -prefMapSize 247458 -appDir /usr/lib/firefox/browser {91008bef-b6d6-452f-b4a6-e78d887b3569} 351306 true
rdd
abhijit 353500 3662 0 Jan06 ? 00:00:00 /usr/libexec/gvfsd-http --spawner :1.3 /org/gtk/gvfs/exec_spaw/4
abhijit 364809 3890 0 Jan06 ? 00:19:20 /usr/lib/virtualbox/VirtualBox
abhijit 364837 3629 0 Jan06 ? 00:16:49 /usr/lib/virtualbox/VBoxXPCOMIPCD
abhijit 364845 3629 0 Jan06 ? 00:29:44 /usr/lib/virtualbox/VBoxSVC --auto-shutdown
root 364957 2 0 Jan06 ? 00:00:00 [dio/dm-0]
abhijit 369749 30565 0 Jan06 pts/3 00:00:00 bash
abhijit 369879 30565 0 Jan06 pts/4 00:00:00 bash
abhijit 379620 30565 0 Jan06 pts/6 00:00:00 bash
abhijit 381917 3890 0 Jan06 ? 00:00:46 flameshot
root 386201 2 0 Jan06 ? 00:00:02 [kworker/0:2H-acpi_thermal_pm]
postfix 389291 3508 0 Jan06 ? 00:00:00 qmgr -l -t unix -u
root 486171 2 0 Jan12 ? 00:00:01 [kworker/0:0H-kblockd]
abhijit 527395 3629 0 Jan12 ? 00:00:49 /usr/bin/gedit --gapplication-service
abhijit 551299 351306 0 Jan12 ? 00:01:54 /usr/lib/firefox/firefox -contentproc -childID 438 -isForBrowser -
prefsLen 35109 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {c4ebd70d-bad7-4e7c-9066-39827103c84e} 351306
true tab
abhijit 552002 3890 0 Jan12 ? 00:03:32 /opt/Signal/signal-desktop --no-sandbox
abhijit 552005 552002 0 Jan12 ? 00:00:00 /opt/Signal/signal-desktop --type=zygote --no-zygote-sandbox --no-
sandbox
abhijit 552006 552002 0 Jan12 ? 00:00:00 /opt/Signal/signal-desktop --type=zygote --no-sandbox
abhijit 552037 552005 0 Jan12 ? 00:03:28 /opt/Signal/signal-desktop --type=gpu-process --no-sandbox --
enable-crash-reporter=18887fa1-4d37-46f0-bf9f-b37513c81cf9,no_channel --user-data-dir=/home/abhijit/.config/Signal --
gpu-
preferences=WAAAAAAAAAagAAAEAAAAAAAAAAAAAAAAABgAAAAAAAA4AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABAAAAGAAAAAAAAAYAAAA
--use-gl=angle --use-angle=swiftshader-webgl --shared-files --field-trial-
handle=0,i,3213315393136307567,15155041764863120512,262144 --disable-
features=HardwareMediaKeyHandling,SpareRendererForSitePerProcess
abhijit 552045 552002 0 Jan12 ? 00:00:04 /opt/Signal/signal-desktop --type=utility --utility-sub-
type=network.mojom.NetworkService --lang=en-GB --service-sandbox-type=none --no-sandbox --enable-crash-
reporter=18887fa1-4d37-46f0-bf9f-b37513c81cf9,no_channel --user-data-dir=/home/abhijit/.config/Signal --shared-
files=v8_context_snapshot_data:100 --field-trial-handle=0,i,3213315393136307567,15155041764863120512,262144 --disable-
features=HardwareMediaKeyHandling,SpareRendererForSitePerProcess
abhijit 552103 552002 1 Jan12 ? 00:57:51 /opt/Signal/signal-desktop --type=renderer --enable-crash-
reporter=18887fa1-4d37-46f0-bf9f-b37513c81cf9,no_channel --user-data-dir=/home/abhijit/.config/Signal --app-
path=/opt/Signal/resources/app.asar --no-sandbox --no-zygote --enable-blink-features=CSSPseudoDir,CSSLogical --disable-
blink-features=Accelerated2dCanvas,AcceleratedSmallCanvases --first-renderer-process --no-sandbox --disable-gpu-
compositing --lang=en-GB --num-raster-threads=4 --enable-main-frame-before-activation --renderer-client-id=4 --time-
ticks-at-unix-epoch=-1704847690836396 --launch-time-ticks=218630770368 --shared-files=v8_context_snapshot_data:100 --
field-trial-handle=0,i,3213315393136307567,15155041764863120512,262144 --disable-
features=HardwareMediaKeyHandling,SpareRendererForSitePerProcess
abhijit 552149 552002 0 Jan12 ? 00:00:12 /opt/Signal/signal-desktop --type=utility --utility-sub-
type=audio.mojom.AudioService --lang=en-GB --service-sandbox-type=none --no-sandbox --enable-crash-reporter=18887fa1-
4d37-46f0-bf9f-b37513c81cf9,no_channel --user-data-dir=/home/abhijit/.config/Signal --shared-
files=v8_context_snapshot_data:100 --field-trial-handle=0,i,3213315393136307567,15155041764863120512,262144 --disable-
features=HardwareMediaKeyHandling,SpareRendererForSitePerProcess
abhijit 554660 351306 0 Jan13 ? 00:04:20 /usr/lib/firefox/firefox -contentproc -childID 442 -isForBrowser -
prefsLen 35109 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {2cffa6f3-6f8c-4a3f-880d-01cc71baf983} 351306
true tab
abhijit 554855 351306 4 Jan13 ? 03:24:06 /usr/lib/firefox/firefox -contentproc -childID 445 -isForBrowser -
prefsLen 35109 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {abcba8e9-5b97-4281-ad64-1356841dcf34} 351306
true tab
abhijit 556349 351306 0 Jan13 ? 00:01:18 /usr/lib/firefox/firefox -contentproc -childID 451 -isForBrowser -
prefsLen 35109 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {6318d453-643c-4b85-9f0f-bfd0a429cd41} 351306
true tab
```



```
abhijit 557600 3629 0 Jan13 ? 00:00:08 /usr/lib/speech-dispatcher-modules/sd_espeak-ng /etc/speech-
dispatcher/modules/espeak-ng.conf
abhijit 557607 3629 0 Jan13 ? 00:00:08 /usr/lib/speech-dispatcher-modules/sd_dummy /etc/speech-
dispatcher/modules/dummy.conf
abhijit 557610 3629 0 Jan13 ? 00:00:08 /usr/lib/speech-dispatcher-modules/sd_generic /etc/speech-
dispatcher/modules/mary-generic.conf
abhijit 557613 3629 0 Jan13 ? 00:00:00 /usr/bin/speech-dispatcher --spawn --communication-method
unix_socket --socket-path /run/user/1000/speech-dispatcher/speechd.sock
abhijit 571320 351306 0 Jan13 ? 00:01:16 /usr/lib/firefox/firefox -contentproc -childID 486 -isForBrowser -
prefsLen 35110 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {71966a59-b1df-4c44-975c-62d7cd41188c} 351306
true tab
abhijit 571410 351306 0 Jan13 ? 00:01:34 /usr/lib/firefox/firefox -contentproc -childID 488 -isForBrowser -
prefsLen 35110 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {bfbcb15ff-6ca5-4837-b97e-334e0bfc8855} 351306
true tab
abhijit 571582 351306 0 Jan13 ? 00:03:32 /usr/lib/firefox/firefox -contentproc -childID 492 -isForBrowser -
prefsLen 35110 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {04e791e5-9cde-4e03-9211-141bdb440139} 351306
true tab
abhijit 591339 4139495 0 Jan13 pts/9 00:00:00 vi mh-pyt.txt
root 594356 2 0 Jan13 ? 00:00:00 [dio/nvme0n1p7]
abhijit 594726 30565 0 Jan13 pts/5 00:00:00 bash
abhijit 791421 351306 1 Jan14 ? 00:40:27 /usr/lib/firefox/firefox -contentproc -childID 839 -isForBrowser -
prefsLen 35227 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {b6f69be3-7ebe-43c9-9d55-c72161180b5e} 351306
true tab
abhijit 794226 351306 0 Jan14 ? 00:02:03 /usr/lib/firefox/firefox -contentproc -childID 848 -isForBrowser -
prefsLen 35227 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {28fddc81-ea39-4496-a1b6-b70d9a8a9c31} 351306
true tab
abhijit 797871 351306 0 Jan14 ? 00:02:41 /usr/lib/firefox/firefox -contentproc -childID 851 -isForBrowser -
prefsLen 35226 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {b254a3c7-e1f7-4245-9fcf-820074a4e69f} 351306
true tab
abhijit 799607 30565 0 Jan14 pts/11 00:00:00 bash
abhijit 803503 30565 0 Jan14 pts/12 00:00:00 bash
abhijit 815513 799607 0 Jan15 pts/11 00:00:00 vi timetable-todo2
abhijit 821738 351306 0 Jan15 ? 00:07:30 /usr/lib/firefox/firefox -contentproc -childID 995 -isForBrowser -
prefsLen 35227 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {90cad852-941e-44b3-8fa4-0d44cbcfda7a} 351306
true tab
abhijit 895193 351306 0 Jan15 ? 00:00:04 /usr/lib/firefox/firefox -contentproc -childID 1131 -isForBrowser -
prefsLen 35227 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {218262bd-4f9e-423a-9368-4d2e44f33125} 351306
true tab
root 942398 1 0 10:40 ? 00:00:00 /usr/sbin/cupsd -l
root 942399 1 0 10:40 ? 00:00:00 /usr/sbin/cups-browsed
www-data 942465 1420 0 10:40 ? 00:00:00 /usr/sbin/apache2 -k start
www-data 942466 1420 0 10:40 ? 00:00:00 /usr/sbin/apache2 -k start
www-data 942468 1420 0 10:40 ? 00:00:00 /usr/sbin/apache2 -k start
www-data 942469 1420 0 10:40 ? 00:00:00 /usr/sbin/apache2 -k start
www-data 942470 1420 0 10:40 ? 00:00:00 /usr/sbin/apache2 -k start
www-data 942471 1420 0 10:40 ? 00:00:00 /usr/sbin/apache2 -k start
abhijit 954109 30565 0 11:21 pts/2 00:00:00 bash
abhijit 961628 351306 3 12:36 ? 00:05:34 /usr/lib/firefox/firefox -contentproc -childID 1704 -isForBrowser -
prefsLen 35227 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {cb82b978-ef9d-4a76-b59c-5b13b652a0ec} 351306
true tab
root 961877 2 0 12:36 ? 00:00:02 [kworker/u32:5-events_unbound]
root 962113 2 0 12:39 ? 00:00:08 [kworker/u33:3-hci0]
abhijit 968060 351306 0 13:09 ? 00:00:05 /usr/lib/firefox/firefox -contentproc -childID 1749 -isForBrowser -
prefsLen 35227 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {c0ba3673-966e-48a6-b029-91346e348342} 351306
true tab
root 968299 2 0 13:11 ? 00:00:02 [kworker/u32:4-events_unbound]
root 969560 2 0 13:23 ? 00:00:01 [kworker/5:1-cgroup_destroy]
root 969608 2 0 13:24 ? 00:00:00 [kworker/u32:7-events_unbound]
root 969648 2 0 13:24 ? 00:00:01 [kworker/2:0-inet_frag_wq]
abhijit 969715 379620 0 13:25 pts/6 00:00:00 ssh root@10.1.101.41
```

root	970435	2	0	13:27 ?	00:00:01 [kworker/3:2-rcu_gp]
root	971921	2	0	13:33 ?	00:00:00 [kworker/1:2-rcu_gp]
root	972048	2	0	13:35 ?	00:00:00 [kworker/7:2-rcu_gp]
root	972127	2	0	13:37 ?	00:00:01 [kworker/0:1-events_long]
root	972207	2	0	13:37 ?	00:00:01 [kworker/4:1-events]
root	972378	2	0	13:39 ?	00:00:00 [kworker/1:0-cgroup_destroy]
root	972435	2	0	13:39 ?	00:00:00 [kworker/6:2-events]
root	972964	2	0	13:41 ?	00:00:00 [kworker/7:0-events]
root	973166	2	0	13:41 ?	00:00:00 [kworker/u32:0-events_unbound]
root	973193	2	0	13:42 ?	00:00:00 [kworker/2:1-events]
root	973282	2	0	13:43 ?	00:00:00 [kworker/4:2-events]
root	973384	2	0	13:44 ?	00:00:00 [kworker/3:0-rcu_gp]
root	973389	2	0	13:44 ?	00:00:01 [kworker/u33:0-hci0]
root	973391	2	0	13:44 ?	00:00:00 [kworker/6:0-rcu_gp]
root	973392	2	0	13:44 ?	00:00:00 [kworker/5:2-events]
root	973655	2	0	13:45 ?	00:00:00 [kworker/1:1-events]
root	973664	2	0	13:46 ?	00:00:00 [kworker/7:1-events]
root	973965	2	0	13:47 ?	00:00:00 [kworker/2:2-events]
root	974126	2	0	13:48 ?	00:00:00 [kworker/u32:1-kcryptd/253:0]
root	974189	2	0	13:48 ?	00:00:00 [kworker/4:0-events]
root	974190	2	0	13:48 ?	00:00:00 [kworker/0:2-events]
root	974369	2	0	13:49 ?	00:00:00 [kworker/u32:2-nvme-wq]
root	974539	2	0	13:49 ?	00:00:00 [kworker/u32:3-events_unbound]
root	974655	2	0	13:50 ?	00:00:00 [kworker/6:1-events]
root	974690	2	0	13:50 ?	00:00:00 [kworker/5:0-events]
root	974740	2	0	13:50 ?	00:00:00 [kworker/7:3-events]
root	974742	2	0	13:50 ?	00:00:00 [kworker/3:1-pm]
root	974743	2	0	13:50 ?	00:00:00 [kworker/u32:6-events_unbound]
root	974744	2	0	13:50 ?	00:00:00 [kworker/u32:8-kcryptd/253:0]
root	974745	2	0	13:50 ?	00:00:00 [kworker/u32:9-events_unbound]
root	974746	2	0	13:50 ?	00:00:00 [kworker/u32:10-events_unbound]
root	974747	2	0	13:50 ?	00:00:00 [kworker/u32:11-events_unbound]
root	974748	2	0	13:50 ?	00:00:00 [kworker/u32:12-kcryptd/253:0]
root	974749	2	0	13:50 ?	00:00:00 [kworker/u32:13-events_unbound]
root	974750	2	0	13:50 ?	00:00:00 [kworker/u32:14-events_unbound]
root	974751	2	0	13:50 ?	00:00:00 [kworker/u32:15-events_unbound]
root	974752	2	0	14:18 ?	00:00:00 [kworker/u32:16-events_unbound]
root	974753	2	0	14:18 ?	00:00:00 [kworker/u32:17-events_unbound]
root	974754	2	0	14:18 ?	00:00:00 [kworker/u32:18-events_unbound]
root	974755	2	0	14:18 ?	00:00:00 [kworker/u32:19-events_unbound]
root	974756	2	0	14:18 ?	00:00:00 [kworker/u32:20-kcryptd/253:0]
root	974757	2	0	14:18 ?	00:00:00 [kworker/u32:21-events_unbound]
root	974758	2	0	14:18 ?	00:00:00 [kworker/u32:22-events_unbound]
root	974759	2	0	14:18 ?	00:00:00 [kworker/u32:23-events_unbound]
root	974760	2	0	14:18 ?	00:00:00 [kworker/u32:24-events_unbound]
root	974761	2	0	14:18 ?	00:00:00 [kworker/u32:25-events_unbound]
root	974762	2	0	14:18 ?	00:00:00 [kworker/u32:26-events_unbound]
root	974763	2	0	14:18 ?	00:00:00 [kworker/u32:27-kcryptd/253:0]
root	974764	2	0	14:18 ?	00:00:00 [kworker/u32:28-kcryptd/253:0]
root	974765	2	0	14:18 ?	00:00:00 [kworker/u32:29-events_unbound]
root	974766	2	0	14:18 ?	00:00:00 [kworker/u32:30+events_unbound]
root	974767	2	0	14:18 ?	00:00:00 [kworker/u32:31-events_unbound]
root	974768	2	0	14:18 ?	00:00:00 [kworker/u32:32-events_unbound]
root	974769	2	0	14:18 ?	00:00:00 [kworker/u32:33-events_unbound]
root	974770	2	0	14:18 ?	00:00:00 [kworker/u32:34-events_unbound]
root	974771	2	0	14:18 ?	00:00:00 [kworker/u32:35-events_unbound]
root	974772	2	0	14:18 ?	00:00:00 [kworker/u32:36-events_unbound]
root	974773	2	0	14:18 ?	00:00:00 [kworker/u32:37-events_unbound]
root	974774	2	0	14:18 ?	00:00:00 [kworker/u32:38-events_unbound]
root	974775	2	0	14:18 ?	00:00:00 [kworker/u32:39-events_unbound]
root	974776	2	0	14:18 ?	00:00:00 [kworker/u32:40-kcryptd/253:0]
root	974778	2	0	14:18 ?	00:00:00 [kworker/u32:42-events_unbound]
root	974779	2	0	14:18 ?	00:00:00 [kworker/3:3-events]
root	974780	2	0	14:18 ?	00:00:00 [kworker/3:4-events]
root	974798	2	0	14:18 ?	00:00:00 [kworker/3:5-events]
root	974995	2	0	14:18 ?	00:00:00 [kworker/u33:2-rb_allocator]
root	975505	2	0	14:20 ?	00:00:00 [kworker/6:3-events]
root	975656	2	0	14:20 ?	00:00:00 [kworker/5:3-events]
root	975657	2	0	14:29 ?	00:00:00 [kworker/1:3-pm]
root	975658	2	0	14:29 ?	00:00:00 [kworker/1:4-events]

```

abhijit  975722  351306  0 14:29 ?          00:00:02 /usr/lib/firefox/firefox -contentproc -childID 1836 -isForBrowser -
prefsLen 35227 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {d9ea12ea-5b3f-477c-9c08-9a4196ea8d04} 351306
true tab
root     976242      2  0 15:16 ?          00:00:00 [kworker/0:0-events]
root     976243      2  0 15:16 ?          00:00:00 [kworker/0:3-events]
root     976244      2  0 15:16 ?          00:00:00 [kworker/0:4-events]
postfix  976248     3508  0 15:16 ?          00:00:00 pickup -l -t unix -u -c
abhijit  976770  351306  0 15:18 ?          00:00:00 /usr/lib/firefox/firefox -contentproc -childID 1840 -isForBrowser -
prefsLen 35227 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {cb567d37-1c03-46db-966c-632f4b708354} 351306
true tab
abhijit  976836  351306  0 15:19 ?          00:00:00 /usr/lib/firefox/firefox -contentproc -childID 1841 -isForBrowser -
prefsLen 35227 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {b62e1986-221e-481e-bda4-62fb37caaa67} 351306
true tab
abhijit  977138  351306  0 15:20 ?          00:00:00 /usr/lib/firefox/firefox -contentproc -childID 1842 -isForBrowser -
prefsLen 35227 -prefMapSize 247458 -jsInitLen 229864 -parentBuildID 20240108143603 -greomni /usr/lib/firefox/omni.ja -
appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib/firefox/browser {989da731-7bd5-44ce-abd0-200ca4c84b9b} 351306
true tab
abhijit  977184  594726  0 15:21 pts/5      00:00:00 ps -eaf
abhijit  1664880     3629  0 Jan07 ?          00:00:06 /usr/bin/gnome-calendar --gapplication-service
abhijit  1665340     3629  0 Jan07 ?          00:00:00 /usr/bin/gpg-agent --supervised
abhijit  3872409     3629  0 Jan07 ?          00:00:05 /usr/bin/seahorse --gapplication-service
root     3873244      1  0 Jan07 ?          00:00:55 /sbin/mount.ntfs /dev/nvme0n1p3 /media/abhijit/windows -o
rw,nodev,nosuid,windows_names,uid=1000,gid=1000,uhelper=udisks2
abhijit  3884359     30565  0 Jan07 pts/7      00:00:00 bash
abhijit  4108623     30565  0 Jan08 pts/8      00:00:00 bash
abhijit  4136834     3890  0 Jan08 ?          00:00:00 /usr/lib/libreoffice/program/oosplash --calc
abhijit  4136869  4136834  0 Jan08 ?          00:32:11 /usr/lib/libreoffice/program/soffice.bin --calc
abhijit  4139495     30565  0 Jan08 pts/9      00:00:00 bash

```

The PID of the grand-parent of the process with PID 4108623 is :



The two processes which were created by kernel have PIDs (in increasing order)

✗ and



The process that created most of the "graphical" processes is having PID



Question **6**

Partially correct

Mark 0.67 out of 1.00

Order the following events in boot process (from 1 onwards)

Shell	<input type="text" value="5"/>	✗
Boot loader	<input type="text" value="2"/>	✓
Init	<input type="text" value="4"/>	✓
BIOS	<input type="text" value="1"/>	✓
Login interface	<input type="text" value="6"/>	✗
OS	<input type="text" value="3"/>	✓

Your answer is partially correct.

You have correctly selected 4.

The correct answer is: Shell → 6, Boot loader → 2, Init → 4, BIOS → 1, Login interface → 5, OS → 3

Question **7**

Correct

Mark 0.50 out of 0.50

Is the terminal a part of the kernel on GNU/Linux systems?

- ☐ a. yes
- ☒ b. no ✓ wrong

The correct answer is: no

Question **8**

Correct

Mark 1.00 out of 1.00

Consider the following programs

`exec1.c`

```
#include <unistd.h>
#include <stdio.h>
int main() {
    execl("./exec2", "./exec2", NULL);
}
```

`exec2.c`

```
#include <unistd.h>
#include <stdio.h>
int main() {
    execl("/bin/ls", "/bin/ls", NULL);

    printf("hello\n");
}
```

Compiled as

`cc exec1.c -o exec1`

`cc exec2.c -o exec2`

And run as

`$./exec1`

Explain the output of the above command (`./exec1`)

Assume that `/bin/ls` , i.e. the 'ls' program exists.

Select one:

- ☐ a. Execution fails as the call to `execl()` in `exec1` fails
- ☐ b. Execution fails as the call to `execl()` in `exec2` fails
- ☐ c. Execution fails as one exec can't invoke another exec
- ☐ d. Program prints hello
- ☒ e. "ls" runs on current directory ✓

Your answer is correct.

The correct answer is: "ls" runs on current directory

Question **9**

Correct

Mark 0.50 out of 0.50

When you turn your computer ON, on BIOS based systems, you are often shown an option like "Press F9 for boot options". What does this mean?

- ☐ a. The choice of booting slowly or fast
- ☒ b. The BIOS allows us to choose the boot device, the device from which the boot loader will be loaded ✓
- ☐ c. The choice of which OS to boot from
- ☐ d. The choice of the boot loader (e.g. GRUB or Windows-Loader)

The correct answer is: The BIOS allows us to choose the boot device, the device from which the boot loader will be loaded

Question **10**

Correct

Mark 1.00 out of 1.00

How does the distinction between kernel mode and user mode function as a rudimentary form of protection (security) ?

Select one:

- ☐ a. It prohibits invocation of kernel code completely, if a user program is running
- ☐ b. It prohibits one process from accessing other process's memory
- ☐ c. It disallows hardware interrupts when a process is running
- ☒ d. It prohibits a user mode process from running privileged instructions ✓

Your answer is correct.

The correct answer is: It prohibits a user mode process from running privileged instructions

Question **11**

Correct

Mark 1.00 out of 1.00

Select all the correct statements about bootloader.

Every wrong selection will deduct marks proportional to $1/n$ where n is total wrong choices in the question.

You will get minimum a zero.

- ☐ a. Bootloader must be one sector in length
- ☐ b. The bootloader loads the BIOS
- ☒ c. Bootloaders allow selection of OS to boot from ✓
- ☒ d. Modern Bootloaders often allow configuring the way an OS boots ✓
- ☒ e. LILO is a bootloader ✓

Your answer is correct.

The correct answers are: LILO is a bootloader, Modern Bootloaders often allow configuring the way an OS boots, Bootloaders allow selection of OS to boot from

Question **12**

Correct

Mark 1.00 out of 1.00

Predict the output of the program given here.

Assume that all the path names for the programs are correct. For example "/usr/bin/echo" will actually run echo command.

Assume that there is no mixing of printf output on screen if two of them run concurrently.

In the answer replace a new line by a single space.

For example::

good

output

should be written as good output

--

```
main() {  
    int i;  
    i = fork();  
    if(i == 0)  
        execl("/usr/bin/echo", "/usr/bin/echo", "hi", 0);  
    else  
        wait(0);  
    fork();  
    execl("/usr/bin/echo", "/usr/bin/echo", "one", 0);  
}
```

Answer: hi one one



The correct answer is: hi one one

Question **13**

Correct

Mark 1.00 out of 1.00

Select all statements that correctly explain the use/purpose of system calls.

Select one or more:

- ☒ a. Provide an environment for process creation ✓
- ☐ b. Handle exceptions like division by zero
- ☒ c. Allow I/O device access to user processes ✓
- ☒ d. Provide services for accessing files ✓
- ☐ e. Handle ALL types of interrupts
- ☐ f. Run each instruction of an application program
- ☒ g. Switch from user mode to kernel mode ✓

Your answer is correct.

The correct answers are: Switch from user mode to kernel mode, Provide services for accessing files, Allow I/O device access to user processes, Provide an environment for process creation

Question **14**

Partially correct

Mark 0.60 out of 1.00

Select all the correct statements about two modes of CPU operation

Select one or more:

- ☐ a. Some instructions are allowed to run only in user mode, while all instructions can run in kernel mode
- ☒ b. There is an instruction like 'iret' to return from kernel mode to user mode ✓
- ☒ c. The two modes are essential for a multiprogramming system ✓
- ☒ d. The two modes are essential for a multitasking system ✓
- ☐ e. The software interrupt instructions change the mode from user mode to kernel mode and jumps to predefined location simultaneously

Your answer is partially correct.

You have correctly selected 3.

The correct answers are: The two modes are essential for a multiprogramming system, The two modes are essential for a multitasking system, There is an instruction like 'iret' to return from kernel mode to user mode, The software interrupt instructions change the mode from user mode to kernel mode and jumps to predefined location simultaneously, Some instructions are allowed to run only in user mode, while all instructions can run in kernel mode

Question **15**

Partially correct

Mark 0.25 out of 0.50

Select all the correct statements about bootloader.

Every wrong selection will deduct marks proportional to $1/n$ where n is total wrong choices in the question.

You will get minimum a zero.

- ☒ a. Bootloaders allow selection of OS to boot from ✓
- ☒ b. Bootloader must be one sector in length ✗
- ☒ c. LILO is a bootloader ✓
- ☐ d. The bootloader loads the BIOS
- ☒ e. Modern Bootloaders often allow configuring the way an OS boots ✓

Your answer is partially correct.

You have selected too many options.

The correct answers are: LILO is a bootloader, Modern Bootloaders often allow configuring the way an OS boots, Bootloaders allow selection of OS to boot from

Question 16

Partially correct

Mark 0.33 out of 1.00

Select the sequence of events that are NOT possible, assuming a non-interruptible kernel code

(Note: non-interruptible kernel code means, if the kernel code is executing, then interrupts will be disabled).

Note: A possible sequence may have some missing steps in between. An impossible sequence will have n and n+1th steps such that n+1th step can not follow n'th step.

Select one or more:

- ☐ a. P1 running
P1 makes sytem call and blocks
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P3 running
Hardware interrupt
Interrupt unblocks P1
Interrupt returns
P3 running
Timer interrupt
Scheduler
P1 running
- ☐ b. P1 running
keyboard hardware interrupt
keyboard interrupt handler running
interrupt handler returns
P1 running
P1 makes sytem call
system call returns
P1 running
timer interrupt
scheduler
P2 running
- ☒ c. P1 running ✓
P1 makes sytem call and blocks
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P1 running again
- ☐ d.
P1 running
P1 makes sytem call
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P1 running again
- ☐ e. P1 running
P1 makes system call
system call returns
P1 running
timer interrupt
Scheduler running
P2 running
- ☐ f. P1 running
P1 makes system call
timer interrupt
Scheduler
P2 running
timer interrupt
Scheuler
P1 running
P1's system call return

Your answer is partially correct.

You have correctly selected 1.

The correct answers are: P1 running

P1 makes sytem call and blocks

Scheduler

P2 running

P2 makes sytem call and blocks

Scheduler

P1 running again, P1 running

P1 makes system call

timer interrupt

Scheduler

P2 running

timer interrupt

Scheuler

P1 running

P1's system call return,

P1 running

P1 makes sytem call

Scheduler

P2 running

P2 makes sytem call and blocks

Scheduler

P1 running again

Question 17

Correct

Mark 1.00 out of 1.00

What will this program do?

```
int main() {  
    fork();  
    execl("/bin/ls", "/bin/ls", NULL);  
    printf("hello");  
}
```

- ☐ a. run ls twice and print hello twice
- ☐ b. run ls twice and print hello twice, but output will appear in some random order
- ☒ c. run ls twice ✓
- ☐ d. one process will run ls, another will print hello
- ☐ e. run ls once

Your answer is correct.

The correct answer is: run ls twice

[◀ Surprise Quiz - 1 \(pre-requisites\)](#)

Jump to...

[Surprise Quiz - 3 \(processes, memory management, event driven kernel\), compilation-linking-loading](#) ►

Started on	Wednesday, 7 February 2024, 6:09 PM
State	Finished
Completed on	Wednesday, 7 February 2024, 7:10 PM
Time taken	1 hour
Grade	18.14 out of 20.00 (90.68%)

Question 1

Correct

Mark 1.00 out of 1.00

Consider the following code and MAP the file to which each fd points at the end of the code. Assume that files/folders exist when needed with proper permissions and open() calls work.

```

int main(int argc, char *argv[]) {
    int fd1, fd2 = 1, fd3 = 1, fd4 = 1;

    fd1 = open("/tmp/1", O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);
    fd2 = open("/tmp/2", O_RDONLY);
    fd3 = open("/tmp/3", O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);
    close(0);
    close(1);
    dup(fd2);
    dup(fd3);
    close(fd3);
    dup2(fd2, fd4);
    printf("%d %d %d %d\n", fd1, fd2, fd3, fd4);
    return 0;
}
    
```

fd2

/tmp/2

✓

fd1

/tmp/1

✓

0

/tmp/2

✓

fd3

closed

✓

2

stderr

✓

1

/tmp/3

✓

fd4

/tmp/2

✓

The correct answer is: fd2 → /tmp/2, fd1 → /tmp/1, 0 → /tmp/2, fd3 → closed, 2 → stderr, 1 → /tmp/3, fd4 → /tmp/2

Question **2**

Partially correct

Mark 1.43 out of 2.00

Order the events that occur on a timer interrupt:

Save the context of the currently running process	3	✓
Jump to scheduler code	4	✓
Set the context of the new process	6	✓
Jump to a code pointed by IDT	1	✗
Change to kernel stack of currently running process	2	✗
Select another process for execution	5	✓
Execute the code of the new process	7	✓

The correct answer is: Save the context of the currently running process → 3, Jump to scheduler code → 4, Set the context of the new process → 6, Jump to a code pointed by IDT → 2, Change to kernel stack of currently running process → 1, Select another process for execution → 5, Execute the code of the new process → 7




Question 3

Partially correct

Mark 0.75 out of 1.00

Select the sequence of events that are NOT possible, assuming an interruptible kernel code

Select one or more:

- ☐ a. P1 running
P1 makes sytem call and blocks
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P3 running
Hardware interrupt
Interrupt unblocks P1
Interrupt returns
P3 running
Timer interrupt
Scheduler
P1 running
- ☐ b. P1 running
keyboard hardware interrupt
keyboard interrupt handler running
interrupt handler returns
P1 running
P1 makes sytem call
system call returns
P1 running
timer interrupt
scheduler
P2 running
- ☒ c. P1 running 
P1 makes system call
timer interrupt
Scheduler
P2 running
timer interrupt
Scheuler
P1 running
P1's system call return
- ☒ d. P1 running 
P1 makes sytem call and blocks
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P1 running again
- ☒ e. 
P1 running
P1 makes sytem call
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P1 running again
- ☐ f. P1 running
P1 makes system call
system call returns

P1 running
timer interrupt
Scheduler running
P2 running

The correct answers are: P1 running

P1 makes sytem call and blocks

Scheduler

P2 running

P2 makes sytem call and blocks

Scheduler

P1 running again,

P1 running

P1 makes sytem call

Scheduler

P2 running

P2 makes sytem call and blocks

Scheduler

P1 running again

Question 4

Partially correct

Mark 1.60 out of 2.00

Match the elements of C program to their place in memory

Malloced Memory	Heap	✓
Global variables	Data	✓
Arguments	Stack	✓
#include files	No Memory needed	✗
Local Static variables	Data	✓
Global Static variables	Data	✓
Function code	Code	✓
Local Variables	Stack	✓
#define MACROS	No Memory needed	✓
Code of main()	Main_Code	✗

The correct answer is: Malloced Memory → Heap, Global variables → Data, Arguments → Stack, #include files → No memory needed, Local Static variables → Data, Global Static variables → Data, Function code → Code, Local Variables → Stack, #define MACROS → No Memory needed, Code of main() → Code

Question **5**

Correct

Mark 1.00 out of 1.00

Select the order in which the various stages of a compiler execute.

Intermediate code generation	<input type="text" value="3"/>	✓
Syntactical Analysis	<input type="text" value="2"/>	✓
Pre-processing	<input type="text" value="1"/>	✓
Linking	<input type="text" value="4"/>	✓
Loading	<input type="text" value="does not exist"/>	✓

The correct answer is: Intermediate code generation → 3, Syntactical Analysis → 2, Pre-processing → 1, Linking → 4, Loading → does not exist

Question 6

Correct

Mark 2.00 out of 2.00

Consider the two programs given below to implement the command (ignore the fact that error checks are not done on return values of functions)

`$ ls ./tmp/asdfksdf >/tmp/ddd 2>&1`

Program 1

```
int main(int argc, char *argv[]) {
    int fd, n, i;
    char buf[128];

    fd = open("/tmp/ddd", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    close(1);
    dup(fd);
    close(2);
    dup(fd);
    execl("/bin/ls", "/bin/ls", ".", "/tmp/asldjfaldfs", NULL);
}
```

Program 2

```
int main(int argc, char *argv[]) {
    int fd, n, i;
    char buf[128];

    close(1);
    fd = open("/tmp/ddd", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    close(2);
    fd = open("/tmp/ddd", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    execl("/bin/ls", "/bin/ls", ".", "/tmp/asldjfaldfs", NULL);
}
```

Select all the correct statements about the programs

Select one or more:

- ☐ a. Program 1 ensures 2>&1 and does not ensure > /tmp/ddd
- ☐ b. Program 1 is correct for > /tmp/ddd but not for 2>&1
- ☒ c. Only Program 1 is correct ✓
- ☐ d. Program 1 does 1>&2
- ☐ e. Program 2 does 1>&2
- ☐ f. Both program 1 and 2 are incorrect
- ☐ g. Program 2 makes sure that there is one file offset used for '2' and '1'
- ☐ h. Program 2 is correct for > /tmp/ddd but not for 2>&1
- ☒ i. Program 1 makes sure that there is one file offset used for '2' and '1' ✓
- ☐ j. Only Program 2 is correct
- ☐ k. Program 2 ensures 2>&1 and does not ensure > /tmp/ddd
- ☐ l. Both programs are correct

The correct answers are: Only Program 1 is correct, Program 1 makes sure that there is one file offset used for '2' and '1'

Question 7

Correct

Mark 1.00 out of 1.00

Select all the correct statements about zombie processes

Select one or more:

- ☒ a. A process can become zombie if it finishes, but the parent has finished before it ✓
- ☒ b. `init()` typically keeps calling `wait()` for zombie processes to get cleaned up ✓
- ☒ c. A process becomes zombie when it finishes, and remains zombie until parent calls `wait()` on it ✓
- ☐ d. Zombie processes are harmless even if OS is up for long time
- ☐ e. A zombie process remains zombie forever, as there is no way to clean it up
- ☒ f. If the parent of a process finishes, before the process itself, then after finishing the process is typically attached to 'init' as parent ✓
- ☐ g. A process becomes zombie when it's parent finishes
- ☒ h. A zombie process occupies space in OS data structures ✓

The correct answers are: A process becomes zombie when it finishes, and remains zombie until parent calls `wait()` on it, A process can become zombie if it finishes, but the parent has finished before it, A zombie process occupies space in OS data structures, If the parent of a process finishes, before the process itself, then after finishing the process is typically attached to 'init' as parent, `init()` typically keeps calling `wait()` for zombie processes to get cleaned up

Question 8

Correct

Mark 1.00 out of 1.00

Consider the image given below, which explains how paging works.

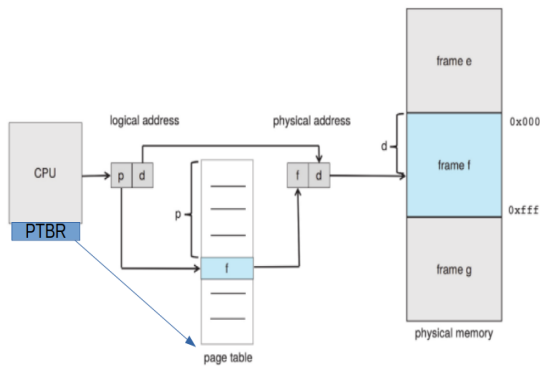


Figure 9.8 Paging hardware.

Mention whether each statement is True or False, with respect to this image.

True	False		
<input checked="" type="radio"/>	<input type="radio"/>	The page table is itself present in Physical memory	✓
<input checked="" type="radio"/>	<input type="radio"/>	Maximum Size of page table is determined by number of bits used for page number	✓
<input checked="" type="radio"/>	<input type="radio"/>	The page table is indexed using page number	✓
<input type="radio"/>	<input checked="" type="radio"/>	The page table is indexed using frame number	✓
<input checked="" type="radio"/>	<input type="radio"/>	The PTBR is present in the CPU as a register	✓
<input checked="" type="radio"/>	<input type="radio"/>	The physical address may not be of the same size (in bits) as the logical address	✓
<input type="radio"/>	<input checked="" type="radio"/>	Size of page table is always determined by the size of RAM	✓
<input type="radio"/>	<input checked="" type="radio"/>	The locating of the page table using PTBR also involves paging translation	✓

The page table is itself present in Physical memory: True

Maximum Size of page table is determined by number of bits used for page number: True

The page table is indexed using page number: True

The page table is indexed using frame number: False

The PTBR is present in the CPU as a register: True

The physical address may not be of the same size (in bits) as the logical address: True

Size of page table is always determined by the size of RAM: False

The locating of the page table using PTBR also involves paging translation: False

Question 9

Correct

Mark 1.00 out of 1.00

Select all the correct statements about MMU and its functionality (on a non-demand paged system)

Select one or more:

- ☐ a. The operating system interacts with MMU for every single address translation
- ☐ b. Illegal memory access is detected by operating system
- ☒ c. Logical to physical address translations in MMU are done in hardware, automatically ✓
- ☐ d. MMU is a separate chip outside the processor
- ☒ e. MMU is inside the processor ✓
- ☒ f. Illegal memory access is detected in hardware by MMU and a trap is raised ✓
- ☒ g. The Operating system sets up relevant CPU registers to enable proper MMU translations ✓
- ☐ h. Logical to physical address translations in MMU are done with specific machine instructions

The correct answers are: MMU is inside the processor, Logical to physical address translations in MMU are done in hardware, automatically, The Operating system sets up relevant CPU registers to enable proper MMU translations, Illegal memory access is detected in hardware by MMU and a trap is raised

Question 10

Partially correct

Mark 0.86 out of 1.00

Mark the statements as True/False w.r.t. the basic concepts of memory management.

True	False		
<input type="radio"/>	<input checked="" type="radio"/>	When a process is executing, each virtual address is converted into physical address by the kernel directly.	✓
<input checked="" type="radio"/>	<input type="radio"/>	The compiler generates address references for code/data/stack/heap in the executable file, depending on the MM architecture provided by CPU and kernel.	✓
<input type="radio"/>	<input checked="" type="radio"/>	The compiler generates the address references for code/data/stack/heap in the executable file as per the memory management schema chosen by the compiler itself, and then the kernel ensures that program is executed with this schema.	✓
<input type="radio"/>	<input checked="" type="radio"/>	The compiler interacts with the kernel continuously while compiling a program and obtains the correct set of memory addresses for code/stack/heap/data and then generates the machine code file.	✓
<input checked="" type="radio"/>	<input type="radio"/>	The kernel ensures that the MMU is setup before scheduling a process and then the CPU/MMU ensures that the address translation takes place.	✓
<input checked="" type="radio"/>	<input type="radio"/>	When a process is executing, each virtual address is converted into physical address by the CPU hardware directly.	✓
<input checked="" type="radio"/>	<input checked="" type="radio"/>	The kernel refers to the page table for converting each virtual address to physical address.	✗

When a process is executing, each virtual address is converted into physical address by the kernel directly.: False

The compiler generates address references for code/data/stack/heap in the executable file, depending on the MM architecture provided by CPU and kernel.: True

The compiler generates the address references for code/data/stack/heap in the executable file as per the memory management schema chosen by the compiler itself, and then the kernel ensures that program is executed with this schema.: False

The compiler interacts with the kernel continuously while compiling a program and obtains the correct set of memory addresses for code/stack/heap/data and then generates the machine code file.: False

The kernel ensures that the MMU is setup before scheduling a process and then the CPU/MMU ensures that the address translation takes place.: True

When a process is executing, each virtual address is converted into physical address by the CPU hardware directly.: True

The kernel refers to the page table for converting each virtual address to physical address.: False

Question **11**

Partially correct

Mark 0.50 out of 1.00

Select the correct statements about paging (not demand paging) mechanism

Select one or more:

- ☐ a. User process can update it's own PTBR
- ☐ b. An invalid entry on a page means, it was an illegal memory reference
- ☒ c. The PTBR is loaded by the OS ✓
- ☐ d. Page table is accessed by the OS as part of execution of an instruction
- ☐ e. User process can update it's own page table entries
- ☒ f. Page table is accessed by the MMU as part of execution of an instruction ✓
- ☒ g. OS creates the page table for every process ✓
- ☒ h. An invalid entry on a page means, either it was illegal memory reference or the page was not present in memory. ✗

The correct answers are: OS creates the page table for every process, The PTBR is loaded by the OS, Page table is accessed by the MMU as part of execution of an instruction, An invalid entry on a page means, it was an illegal memory reference

Question **12**

Correct

Mark 1.00 out of 1.00

Select the compiler's view of the process's address space, for each of the following MMU schemes:
(Assume that each scheme,e.g. paging/segmentation/etc is effectively utilised)

Segmentation	many continuous chunks of variable size	✓
Segmentation, then paging	many continuous chunks of variable size	✓
Relocation + Limit	one continuous chunk	✓
Paging	one continuous chunk	✓

The correct answer is: Segmentation → many continuous chunks of variable size, Segmentation, then paging → many continuous chunks of variable size, Relocation + Limit → one continuous chunk, Paging → one continuous chunk

Question **13**

Correct

Mark 1.00 out of 1.00

Select the state that is not possible after the given state, for a process:

- New: Running ✓
- Ready : Waiting ✓
- Running: : None of these ✓
- Waiting: Running ✓

Question **14**

Correct

Mark 1.00 out of 1.00

A process blocks itself means

- ☒ a. The kernel code of system call, called by the process, moves the process to a waiting queue and calls scheduler ✓
- ☐ b. The application code calls the scheduler
- ☐ c. The kernel code of an interrupt handler, moves the process to a waiting queue and calls scheduler
- ☐ d. The kernel code of system call calls scheduler

The correct answer is: The kernel code of system call, called by the process, moves the process to a waiting queue and calls scheduler

Question **15**

Correct

Mark 1.00 out of 1.00

which of the following is not a difference between real mode and protected mode

- ☐ a. in real mode the segment is multiplied by 16, in protected mode segment is used as index in GDT
- ☐ b. in real mode the addressable memory is less than in protected mode
- ☐ c. in real mode general purpose registers are 16 bit, in protected mode they are 32 bit
- ☐ d. processor starts in real mode
- ☒ e. in real mode the addressable memory is more than in protected mode ✓

The correct answer is: in real mode the addressable memory is more than in protected mode

Question **16**

Correct

Mark 1.00 out of 1.00

Predict the output of the program given here.

Assume that there is no mixing of printf output on screen if two of them run concurrently.

In the answer replace a new line by a single space.

For example::

good

output

should be written as good output

--

```
int main() {  
    int pid;  
    printf("hi\n");  
    pid = fork();  
    if(pid == 0) {  
        exit(0);  
    }  
    printf("bye\n");  
    fork();  
    printf("ok\n");  
}
```

Answer:



The correct answer is: hi bye ok ok

Question **17**

Correct

Mark 1.00 out of 1.00

Which of the following are NOT a part of job of a typical compiler?

- ☐ a. Invoke the linker to link the function calls with their code, extern globals with their declaration
- ☐ b. Check the program for syntactical errors
- ☐ c. Convert high level language code to machine code
- ☒ d. Suggest alternative pieces of code that can be written ✓
- ☒ e. Check the program for logical errors ✓
- ☐ f. Process the # directives in a C program

The correct answers are: Check the program for logical errors, Suggest alternative pieces of code that can be written

[◀ Surprise Quiz - 2](#)

Jump to...

[Questions for test on kalloc/kfree/kvmmalloc, etc. ►](#)

[Dashboard](#) / [My courses](#) / [Computer Engineering & IT](#) / [CEIT-Even-sem-20-21](#) / [OS-Even-sem-2020-21](#) / 14 February - 20 February
/ [Quiz-1](#)

Started on Saturday, 20 February 2021, 2:51 PM

State Finished

Completed on Saturday, 20 February 2021, 3:55 PM

Time taken 1 hour 3 mins

Grade 7.30 out of 20.00 (37%)

Question 1

Partially correct

Mark 0.80 out of 1.00

Select all the correct statements about the state of a process.

- ☒ a. A process can self-terminate only when it's running ✓
- ☒ b. Typically, it's represented as a number in the PCB ✓
- ☒ c. A process that is running is not on the ready queue ✓
- ☒ d. Processes in the ready queue are in the ready state ✓
- ☐ e. It is not maintained in the data structures by kernel, it is only for conceptual understanding of programmers
- ☒ f. Changing from running state to waiting state results in "giving up the CPU" ✓
- ☐ g. A process in ready state is ready to receive interrupts
- ☒ h. A waiting process starts running after the wait is over ✗
- ☒ i. A process changes from running to ready state on a timer interrupt ✓
- ☒ j. A process in ready state is ready to be scheduled ✓
- ☒ k. A running process may terminate, or go to wait or become ready again ✓
- ☒ l. A process waiting for I/O completion is typically woken up by the particular interrupt handler code ✓
- ☐ m. A process waiting for any condition is woken up by another process only
- ☐ n. A process changes from running to ready state on a timer interrupt or any I/O wait

Your answer is partially correct.

You have selected too many options.

The correct answers are: Typically, it's represented as a number in the PCB, A process in ready state is ready to be scheduled, Processes in the ready queue are in the ready state, A process that is running is not on the ready queue, A running process may terminate, or go to wait or become ready again, A process changes from running to ready state on a timer interrupt, Changing from running state to waiting state results in "giving up the CPU", A process can self-terminate only when it's running, A process waiting for I/O completion is typically woken up by the particular interrupt handler code

Question 2

Incorrect

Mark 0.00 out of 1.00

For each line of code mentioned on the left side, select the location of sp/esp that is in use

<code>jmp *%eax</code> in <code>entry.S</code>	0x7c00 to 0x10000	✗
<code>ljmp \$(SEG_KCODE<<3), \$start32</code> in <code>bootasm.S</code>	0x10000 to 0x7c00	✗
<code>call bootmain</code> in <code>bootasm.S</code>	0x7c00 to 0x10000	✗
<code>cli</code> in <code>bootasm.S</code>	0x7c00 to 0	✗
<code>readseg((uchar*)elf, 4096, 0);</code> in <code>bootmain.c</code>	The 4KB area in kernel image, loaded in memory, named as 'stack'	✗

Your answer is incorrect.

The correct answer is: `jmp *%eax`

in `entry.S` → The 4KB area in kernel image, loaded in memory, named as 'stack', `ljmp $(SEG_KCODE<<3), $start32`

in `bootasm.S` → Immaterial as the stack is not used here, `call bootmain`

in `bootasm.S` → 0x7c00 to 0, `cli`

in `bootasm.S` → Immaterial as the stack is not used here, `readseg((uchar*)elf, 4096, 0);`

in `bootmain.c` → 0x7c00 to 0

Question 3

Correct

Mark 0.25 out of 0.25

Order the following events in boot process (from 1 onwards)

Boot loader	2	✓
Shell	6	✓
BIOS	1	✓
OS	3	✓
Init	4	✓
Login interface	5	✓

Your answer is correct.

The correct answer is: Boot loader → 2, Shell → 6, BIOS → 1, OS → 3, Init → 4, Login interface → 5

Question 4

Partially correct

Mark 0.30 out of 0.50

Consider the following command and it's output:

```
$ ls -lht xv6.img kernel
-rw-rw-r-- 1 abhijit abhijit 4.9M Feb 15 11:09 xv6.img
-rwxrwxr-x 1 abhijit abhijit 209K Feb 15 11:09 kernel*
```

Following code in bootmain()

```
readseg((uchar*)elf, 4096, 0);
```

and following selected lines from Makefile

```
xv6.img: bootblock kernel
    dd if=/dev/zero of=xv6.img count=10000
    dd if=bootblock of=xv6.img conv=notrunc
    dd if=kernel of=xv6.img seek=1 conv=notrunc
```

```
kernel: $(OBJJS) entry.o entryother initcode kernel.ld
    $(LD) $(LDFLAGS) -T kernel.ld -o kernel entry.o $(OBJJS) -b binary initcode entryother
    $(OBJDUMP) -S kernel > kernel.asm
    $(OBJDUMP) -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$$/d' > kernel.sym
```

Also read the code of bootmain() in xv6 kernel.

Select the options that describe the meaning of these lines and their correlation.

- ☐ a. Although the size of the kernel file is 209 Kb, only 4Kb out of it is the actual kernel code and remaining part is all zeroes.
- ☒ b. The kernel is compiled by linking multiple .o files created from .c files; and the entry.o, initcode, entryother files ✓
- ☒ c. The kernel.ld file contains instructions to the linker to link the kernel properly ✓
- ☐ d. The bootmain() code does not read the kernel completely in memory
- ☐ e. readseg() reads first 4k bytes of kernel in memory
- ☐ f. Although the size of the xv6.img file is ~5MB, only some part out of it is the bootloader+kernel code and remaining part is all zeroes.
- ☐ g. The kernel.asm file is the final kernel file
- ☐ h. The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is not read as it is user programs.
- ☒ i. The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is read using program headers in bootmain(). ✓

Your answer is partially correct.

You have correctly selected 3.

The correct answers are: The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is read using program headers in bootmain()., readseg() reads first 4k bytes of kernel in memory, The kernel is compiled by linking multiple .o files created from .c files; and the entry.o, initcode, entryother files, The kernel.ld file contains instructions to the linker to link the kernel properly, Although the size of the xv6.img file is ~5MB, only some part out of it is the bootloader+kernel code and remaining part is all zeroes.

Question **5**

Partially correct

Mark 0.50 out of 1.00

```
int f() {  
    int count;  
    for (count = 0; count < 2; count++) {  
        if (fork() == 0)  
            printf("Operating-System\n");  
    }  
    printf("TYCOMP\n");  
}
```

The number of times "Operating-System" is printed, is:

Answer: ☒

The correct answer is: 7.00

Question 6

Partially correct

Mark 0.40 out of 0.50

Select Yes/True if the mentioned element must be a part of PCB

Select No/False otherwise.

Yes	No		
<input checked="" type="radio"/>	<input type="radio"/>	PID	✓
<input checked="" type="radio"/>	<input type="radio"/>	Process context	✓
<input checked="" type="radio"/>	<input type="radio"/>	List of opened files	✓
<input checked="" type="radio"/>	<input type="radio"/>	Process state	✓
<input type="radio"/>	<input checked="" type="radio"/>	Parent's PID	✗
<input type="radio"/>	<input checked="" type="radio"/>	Pointer to IDT	✓
<input type="radio"/>	<input checked="" type="radio"/>	Function pointers to all system calls	✓
<input checked="" type="radio"/>	<input type="radio"/>	Memory management information about that process	✓
<input checked="" type="radio"/>	<input type="radio"/>	Pointer to the parent process	✗
<input checked="" type="radio"/>	<input type="radio"/>	EIP at the time of context switch	✓

PID: Yes

Process context: Yes

List of opened files: Yes

Process state: Yes

Parent's PID: No

Pointer to IDT: No

Function pointers to all system calls: No

Memory management information about that process: Yes

Pointer to the parent process: Yes

EIP at the time of context switch: Yes

Question 7

Incorrect

Mark 0.00 out of 1.00

Select all the correct statements about code of bootmain() in xv6

```
void
bootmain(void)
{
    struct elfhdr *elf;
    struct proghdr *ph, *eph;
    void (*entry)(void);
    uchar* pa;

    elf = (struct elfhdr*)0x10000; // scratch space

    // Read 1st page off disk
    readseg((uchar*)elf, 4096, 0);

    // Is this an ELF executable?
    if(elf->magic != ELF_MAGIC)
        return; // let bootasm.S handle error

    // Load each program segment (ignores ph flags).
    ph = (struct proghdr*)((uchar*)elf + elf->phoff);
    eph = ph + elf->phnum;
    for(; ph < eph; ph++){
        pa = (uchar*)ph->paddr;
        readseg(pa, ph->filesz, ph->off);
        if(ph->memsz > ph->filesz)
            stosb(pa + ph->filesz, 0, ph->memsz - ph->filesz);
    }

    // Call the entry point from the ELF header.
    // Does not return!
    entry = (void(*) (void))(elf->entry);
    entry();
}
```

Also, inspect the relevant parts of the xv6 code. binary files, etc and run commands as you deem fit to answer this question.

- ☒ a. The kernel file gets loaded at the Physical address 0x10000 + 0x80000000 in memory. ✗
- ☒ b. The elf->entry is set by the linker in the kernel file and it's 0x80000000 ✗
- ☒ c. The kernel ELF file contains actual physical address where particular sections of 'kernel' file should be loaded ✓
- ☒ d. The kernel file in memory is not necessarily a continuously filled in chunk, it may have holes in it. ✓
- ☒ e. The kernel file has only two program headers ✓
- ☒ f. The elf->entry is set by the linker in the kernel file and it's 0x80000000 ✗
- ☒ g. The readseg finally invokes the disk I/O code using assembly instructions ✓
- ☒ h. The elf->entry is set by the linker in the kernel file and it's 8010000c ✓
- ☒ i. The kernel file gets loaded at the Physical address 0x10000 in memory. ✓
- ☒ j. The condition if(ph->memsz > ph->filesz) is never true. ✗
- ☒ k. The stosb() is used here, to fill in some space in memory with zeroes ✓

Your answer is incorrect.

The correct answers are: The kernel file gets loaded at the Physical address 0x10000 in memory., The kernel file in memory is not necessarily a continuously filled in chunk, it may have holes in it., The elf->entry is set by the linker in the kernel file and it's 8010000c, The readseg finally invokes the disk I/O code using assembly instructions, The stosb() is used here, to fill in some space in memory with zeroes, The kernel ELF file contains actual physical address where particular sections of 'kernel' file should be loaded, The kernel file has only two program headers

Question 8

Partially correct

Mark 0.13 out of 0.25

Which of the following are NOT a part of job of a typical compiler?

- ☒ a. Check the program for logical errors
- ☐ b. Convert high level language code to machine code
- ☐ c. Process the # directives in a C program
- ☐ d. Invoke the linker to link the function calls with their code, extern globals with their declaration
- ☐ e. Check the program for syntactical errors
- ☐ f. Suggest alternative pieces of code that can be written



Your answer is partially correct.

You have correctly selected 1.

The correct answers are: Check the program for logical errors, Suggest alternative pieces of code that can be written

Question 9

Correct

Mark 0.25 out of 0.25

Rank the following storage systems from slowest (first) to fastest(last)

Cache	6	✓
Hard Disk	3	✓
RAM	5	✓
Optical Disks	2	✓
Non volatile memory	4	✓
Registers	7	✓
Magnetic Tapes	1	✓

Your answer is correct.

The correct answer is: Cache → 6, Hard Disk → 3, RAM → 5, Optical Disks → 2, Non volatile memory → 4, Registers → 7, Magnetic Tapes → 1

Question **10**

Partially correct

Mark 0.21 out of 0.50

Which of the following parts of a C program do not have any corresponding machine code ?

- ☐ a. local variable declaration
- ☐ b. global variables
- ☒ c. function calls
- ☒ d. #directives
- ☐ e. expressions
- ☐ f. pointer dereference
- ☒ g. typedefs

✗

✓

✓

Your answer is partially correct.

You have correctly selected 2.

The correct answers are: #directives, typedefs, global variables

Question **11**

Correct

Mark 0.25 out of 0.25

Match a system call with it's description

pipe	create an unnamed FIFO storage with 2 ends - one for reading and another for writing	✓
dup	create a copy of the specified file descriptor into smallest available file descriptor	✓
dup2	create a copy of the specified file descriptor into another specified file descriptor	✓
exec	execute a binary file overlaying the image of current process	✓
fork	create an identical child process	✓

Your answer is correct.

The correct answer is: pipe → create an unnamed FIFO storage with 2 ends - one for reading and another for writing, dup → create a copy of the specified file descriptor into smallest available file descriptor, dup2 → create a copy of the specified file descriptor into another specified file descriptor, exec → execute a binary file overlaying the image of current process, fork → create an identical child process

Question **12**

Correct

Mark 0.25 out of 0.25

Match the register with the segment used with it.

eip	cs	✓
edi	es	✓
esi	ds	✓
ebp	ss	✓
esp	ss	✓

Your answer is correct.

The correct answer is: eip → cs, edi → es, esi → ds, ebp → ss, esp → ss

Question **13**

Correct

Mark 0.25 out of 0.25

What's the trapframe in xv6?

- ☐ a. A frame of memory that contains all the trap handler code
- ☐ b. The sequence of values, including saved registers, constructed on the stack when an interrupt occurs, built by hardware only
- ☐ c. The IDT table
- ☐ d. A frame of memory that contains all the trap handler code's function pointers
- ☐ e. A frame of memory that contains all the trap handler's addresses
- ☒ f. The sequence of values, including saved registers, constructed on the stack when an interrupt occurs, built by hardware + code in trapasm.S ✓
- ☐ g. The sequence of values, including saved registers, constructed on the stack when an interrupt occurs, built by code in trapasm.S only

Your answer is correct.

The correct answer is: The sequence of values, including saved registers, constructed on the stack when an interrupt occurs, built by hardware + code in trapasm.S

Question 14

Incorrect

Mark 0.00 out of 0.50

Select all the correct statements about linking and loading.

Select one or more:

- ☒ a. Continuous memory management schemes can support dynamic linking and dynamic loading. ✗
- ☒ b. Loader is last stage of the linker program ✗
- ☒ c. Continuous memory management schemes can support static linking and dynamic loading. (may be inefficiently) ✓
- ☒ d. Dynamic linking and loading is not possible without demand paging or demand segmentation. ✓
- ☒ e. Dynamic linking essentially results in relocatable code. ✓
- ☒ f. Continuous memory management schemes can support static linking and static loading. (may be inefficiently) ✓
- ☒ g. Loader is part of the operating system ✓
- ☒ h. Static linking leads to non-relocatable code ✗
- ☒ i. Dynamic linking is possible with continuous memory management, but variable sized partitions only. ✗

Your answer is incorrect.

The correct answers are: Continuous memory management schemes can support static linking and static loading. (may be inefficiently), Continuous memory management schemes can support static linking and dynamic loading. (may be inefficiently), Dynamic linking essentially results in relocatable code., Loader is part of the operating system, Dynamic linking and loading is not possible without demand paging or demand segmentation.

Question 15

Incorrect

Mark 0.00 out of 0.25

In bootasm.S, on the line

```
ljmp $(SEG_KCODE<<3), $start32
```

The SEG_KCODE << 3, that is shifting of 1 by 3 bits is done because

- ☐ a. The value 8 is stored in code segment
- ☐ b. The code segment is 16 bit and only upper 13 bits are used for segment number
- ☒ c. The code segment is 16 bit and only lower 13 bits are used for segment number ✗
- ☐ d. While indexing the GDT using CS, the value in CS is always divided by 8
- ☐ e. The ljmp instruction does a divide by 8 on the first argument

Your answer is incorrect.

The correct answer is: The code segment is 16 bit and only upper 13 bits are used for segment number

Question **16**

Partially correct

Mark 0.07 out of 0.50

Order the events that occur on a timer interrupt:

Change to kernel stack	1	✗
Jump to a code pointed by IDT	2	✗
Jump to scheduler code	5	✗
Set the context of the new process	4	✗
Save the context of the currently running process	3	✓
Execute the code of the new process	6	✗
Select another process for execution	7	✗

Your answer is partially correct.

You have correctly selected 1.

The correct answer is: Change to kernel stack → 2, Jump to a code pointed by IDT → 1, Jump to scheduler code → 4, Set the context of the new process → 6, Save the context of the currently running process → 3, Execute the code of the new process → 7, Select another process for execution → 5

Question **17**

Incorrect

Mark 0.00 out of 1.00

Consider the two programs given below to implement the command (ignore the fact that error checks are not done on return values of functions)

\$ ls . /tmp/asdfksdf >/tmp/ddd 2>&1

Program 1

```
int main(int argc, char *argv[]) {
    int fd, n, i;
    char buf[128];

    fd = open("/tmp/ddd", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    close(1);
    dup(fd);
    close(2);
    dup(fd);
    execl("/bin/ls", "/bin/ls", ".", "/tmp/asldjfaldfs", NULL);
}
```

Program 2

```
int main(int argc, char *argv[]) {
    int fd, n, i;
    char buf[128];

    close(1);
    fd = open("/tmp/ddd", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    close(2);
    fd = open("/tmp/ddd", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    execl("/bin/ls", "/bin/ls", ".", "/tmp/asldjfaldfs", NULL);
}
```

Select all the correct statements about the programs

Select one or more:

- ☒ a. Both programs are correct ✗
- ☒ b. Program 2 makes sure that there is one file offset used for '2' and '1' ✗
- ☒ c. Only Program 2 is correct ✗
- ☒ d. Program 2 does 1>&2 ✗
- ☒ e. Program 2 ensures 2>&1 and does not ensure > /tmp/ddd ✗
- ☒ f. Program 1 makes sure that there is one file offset used for '2' and '1' ✓
- ☒ g. Program 1 is correct for > /tmp/ddd but not for 2>&1 ✗
- ☒ h. Program 1 does 1>&2 ✗
- ☒ i. Both program 1 and 2 are incorrect ✗
- ☒ j. Program 2 is correct for > /tmp/ddd but not for 2>&1 ✗
- ☒ k. Only Program 1 is correct ✓
- ☒ l. Program 1 ensures 2>&1 and does not ensure > /tmp/ddd ✗

Your answer is incorrect.

The correct answers are: Only Program 1 is correct, Program 1 makes sure that there is one file offset used for '2' and '1'



Question **18**

Correct

Mark 0.25 out of 0.25

Select the option which best describes what the CPU does during it's powered ON lifetime

- ☐ a. Ask the user what is to be done, and execute that task
- ☐ b. Ask the OS what is to be done, and execute that task
- ☐ c. Fetch instructions specified by location given by PC, Decode and Execute it, during execution increment PC or change PC as per the instruction itself, Ask the User or the OS what is to be done next, repeat
- ☒ d. Fetch instructions specified by location given by PC, Decode and Execute it, during execution increment PC or change PC as per the instruction itself, repeat ✓
- ☐ e. Fetch instruction specified by OS, Decode and execute it, repeat
- ☐ f. Fetch instructions specified by location given by PC, Decode and Execute it, during execution increment PC or change PC as per the instruction itself, Ask OS what is to be done next, repeat

The correct answer is: Fetch instructions specified by location given by PC, Decode and Execute it, during execution increment PC or change PC as per the instruction itself, repeat

Question 19

Partially correct

Mark 0.86 out of 1.00

Consider the following code and MAP the file to which each fd points at the end of the code.

```
int main(int argc, char *argv[]) {  
    int fd1, fd2 = 1, fd3 = 1, fd4 = 1;  
  
    fd1 = open("/tmp/1", O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);  
    fd2 = open("/tmp/2", O_RDONLY);  
    fd3 = open("/tmp/3", O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);  
    close(0);  
    close(1);  
    dup(fd2);  
    dup(fd3);  
    close(fd3);  
    dup2(fd2, fd4);  
    printf("%d %d %d %d\n", fd1, fd2, fd3, fd4);  
    return 0;  
}
```

1	<input type="text" value="closed"/>	✖
fd4	<input type="text" value="/tmp/2"/>	✔
fd2	<input type="text" value="/tmp/2"/>	✔
fd1	<input type="text" value="/tmp/1"/>	✔
2	<input type="text" value="stderr"/>	✔
0	<input type="text" value="/tmp/2"/>	✔
fd3	<input type="text" value="closed"/>	✔

Your answer is partially correct.

You have correctly selected 6.

The correct answer is: 1 → /tmp/3, fd4 → /tmp/2, fd2 → /tmp/2, fd1 → /tmp/1, 2 → stderr, 0 → /tmp/2, fd3 → closed

Question **20**

Incorrect

Mark 0.00 out of 2.00

Following code claims to implement the command

```
/bin/ls -l | /usr/bin/head -3 | /usr/bin/tail -1
```

Fill in the blanks to make the code work.

Note: Do not include space in writing any option. `x[1][2]` should be written without any space, and so is the case with `[1]` or `[2]`. Pay attention to exact syntax and do not write any extra character like `'` or `=` etc.

```
int main(int argc, char *argv[]) {
```

```
    int pid1, pid2;
```

```
    int pfd[
```

✖ `][2];`

```
    pipe(
```

✖ `);`

```
    pid1 =
```

✖ `;`

```
    if(pid1 != 0) {
```

```
        close(pfd[0]
```

✖ `);`

```
        close(
```

✖ `);`

```
        dup(
```

✖ `);`

```
        execl("/bin/ls", "/bin/ls", "
```

✖ `", NULL);`

```
    }
```

```
    pipe(
```

✖ `);`

✖ `= fork();`

```
    if(pid2 == 0) {
```

```
        close(
```

✖ `;`

```
        close(0);
```

```
        dup(
```

✖ `);`

```
        close(pfd[1]
```

```
    × );  
        close(  
              
        × );  
        dup(  
              
        × );  
        execl("/usr/bin/head", "/usr/bin/head", "  
              
        × ", NULL);  
    } else {  
        close(pfd  
              
        × );  
        close(  
              
        × );  
        dup(  
              
        × );  
        close(pfd  
              
        × );  
        execl("/usr/bin/tail", "/usr/bin/tail", "  
              
        × ", NULL);  
    }  
}
```


Question **21**

Partially correct

Mark 0.11 out of 1.00

Select all the correct statements about calling convention on x86 32-bit.

- ☒ a. Return address is one location above the ebp ✓
- ☒ b. Parameters may be passed in registers or on stack ✓
- ☒ c. Space for local variables is allocated by subtracting the stack pointer inside the code of the called function ✓
- ☒ d. The ebp pointers saved on the stack constitute a chain of activation records ✓
- ☒ e. The two lines in the beginning of each function, "push %ebp; mov %esp, %ebp", create space for local variables ✗
- ☒ f. Parameters may be passed in registers or on stack ✓
- ☒ g. The return value is either stored on the stack or returned in the eax register ✗
- ☐ h. Parameters are pushed on the stack in left-right order
- ☐ i. during execution of a function, ebp is pointing to the old ebp
- ☒ j. Space for local variables is allocated by subtracting the stack pointer inside the code of the caller function ✗
- ☒ k. Compiler may allocate more memory on stack than needed ✓

Your answer is partially correct.

You have selected too many options.

The correct answers are: Compiler may allocate more memory on stack than needed, Parameters may be passed in registers or on stack, Parameters may be passed in registers or on stack, Return address is one location above the ebp, during execution of a function, ebp is pointing to the old ebp, Space for local variables is allocated by subtracting the stack pointer inside the code of the called function, The ebp pointers saved on the stack constitute a chain of activation records

Question **22**

Correct

Mark 1.00 out of 1.00

Match the program with its output (ignore newlines in the output. Just focus on the count of the number of 'hi')

- | | | |
|---|-------|---|
| <code>main() { int i = fork(); if(i == 0) execl("/usr/bin/echo", "/usr/bin/echo", "hi\n", NULL); }</code> | hi | ✓ |
| <code>main() { fork(); execl("/usr/bin/echo", "/usr/bin/echo", "hi\n", NULL); }</code> | hi hi | ✓ |
| <code>main() { int i = NULL; fork(); printf("hi\n"); }</code> | hi hi | ✓ |
| <code>main() { execl("/usr/bin/echo", "/usr/bin/echo", "hi\n", NULL); }</code> | hi | ✓ |

Your answer is correct.

The correct answer is: `main() { int i = fork(); if(i == 0) execl("/usr/bin/echo", "/usr/bin/echo", "hi\n", NULL); }` → hi, `main() { fork(); execl("/usr/bin/echo", "/usr/bin/echo", "hi\n", NULL); }` → hi hi, `main() { int i = NULL; fork(); printf("hi\n"); }` → hi hi, `main() { execl("/usr/bin/echo", "/usr/bin/echo", "hi\n", NULL); }` → hi

Question **23**

Incorrect

Mark 0.00 out of 0.50

Some part of the bootloader of xv6 is written in assembly while some part is written in C. Why is that so?
Select all the appropriate choices

- ☒ a. The code in assembly is required for transition to protected mode, from real mode; but calling convention was applicable all the time ✗
- ☒ b. The setting up of the most essential memory management infrastructure needs assembly code ✓
- ☒ c. The code for reading ELF file can not be written in assembly ✗
- ☒ d. The code in assembly is required for transition to protected mode, from real mode; after that calling convention applies, hence code can be written in C ✓

Your answer is incorrect.

The correct answers are: The code in assembly is required for transition to protected mode, from real mode; after that calling convention applies, hence code can be written in C, The setting up of the most essential memory management infrastructure needs assembly code

Question **24**

Incorrect

Mark 0.00 out of 0.50

xv6.img: bootblock kernel

```
dd if=/dev/zero of=xv6.img count=10000
dd if=bootblock of=xv6.img conv=notrunc
dd if=kernel of=xv6.img seek=1 conv=notrunc
```

Consider above lines from the Makefile. Which of the following is incorrect?

- ☒ a. The size of the kernel file is nearly 5 MB ✓
- ☒ b. The kernel is located at block-1 of the xv6.img ✗
- ☒ c. The xv6.img is of the size 10,000 blocks of 512 bytes each and occupies 10,000 blocks on the disk. ✗
- ☐ d. The size of xv6.img is exactly = (size of bootblock) + (size of kernel)
- ☒ e. The bootblock is located on block-0 of the xv6.img ✗
- ☒ f. The xv6.img is of the size 10,000 blocks of 512 bytes each and occupies upto 10,000 blocks on the disk. ✓
- ☒ g. The bootblock may be 512 bytes or less (looking at the Makefile instruction) ✗
- ☒ h. The xv6.img is the virtual disk that is created by combining the bootblock and the kernel file. ✗
- ☒ i. The size of the xv6.img is nearly 5 MB ✗
- ☒ j. xv6.img is the virtual processor used by the qemu emulator ✓
- ☒ k. Blocks in xv6.img after kernel may be all zeroes. ✗

Your answer is incorrect.

The correct answers are: xv6.img is the virtual processor used by the qemu emulator, The xv6.img is of the size 10,000 blocks of 512 bytes each and occupies upto 10,000 blocks on the disk., The size of the kernel file is nearly 5 MB, The size of xv6.img is exactly = (size of bootblock) + (size of kernel)

Question **25**

Incorrect

Mark 0.00 out of 1.00

Select the sequence of events that are NOT possible, assuming a non-interruptible kernel code

Select one or more:

- ☐ a. P1 running
P1 makes system call
timer interrupt
Scheduler
P2 running
timer interrupt
Scheduler
P1 running
P1's system call return
- ☒ b. P1 running ✓
P1 makes system call and blocks
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P1 running again
- ☐ c. P1 running
P1 makes system call
system call returns
P1 running
timer interrupt
Scheduler running
P2 running
- ☒ d. P1 running ✗
P1 makes system call and blocks
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P3 running
Hardware interrupt
Interrupt unblocks P1
Interrupt returns
P3 running
Timer interrupt
Scheduler
P1 running
- ☐ e.
P1 running
P1 makes system call
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P1 running again
- ☒ f. P1 running ✗
keyboard hardware interrupt
keyboard interrupt handler running
interrupt handler returns
P1 running
P1 makes system call
system call returns

P1 running
timer interrupt
scheduler
P2 running

Your answer is incorrect.

The correct answers are: P1 running

P1 makes system call and blocks

Scheduler

P2 running

P2 makes system call and blocks

Scheduler

P1 running again, P1 running

P1 makes system call

timer interrupt

Scheduler

P2 running

timer interrupt

Scheduler

P1 running

P1's system call return,

P1 running

P1 makes system call

Scheduler

P2 running

P2 makes system call and blocks

Scheduler

P1 running again

Question **26**

Correct

Mark 0.25 out of 0.25

Which of the following are the files related to bootloader in xv6?

- ☐ a. bootasm.s and entry.S
- ☒ b. bootasm.S and bootmain.c
- ☐ c. bootasm.S, bootmain.c and bootblock.c
- ☐ d. bootmain.c and bootblock.S



Your answer is correct.

The correct answer is: bootasm.S and bootmain.c

Question **27**

Correct

Mark 0.25 out of 0.25

Match the following parts of a C program to the layout of the process in memory

Instructions	Text section	✓
Local Variables	Stack Section	✓
Dynamically allocated memory	Heap Section	✓
Global and static data	Data section	✓

Your answer is correct.

The correct answer is:

Instructions → Text section, Local Variables → Stack Section,
Dynamically allocated memory → Heap Section,
Global and static data → Data section

Question **28**

Incorrect

Mark 0.00 out of 0.50

What will this program do?

```
int main() {  
    fork();  
    execl("/bin/ls", "/bin/ls", NULL);  
    printf("hello");  
}
```

- ☐ a. one process will run ls, another will print hello
- ☒ b. run ls once
- ☐ c. run ls twice
- ☐ d. run ls twice and print hello twice
- ☐ e. run ls twice and print hello twice, but output will appear in some random order

✗

Your answer is incorrect.

The correct answer is: run ls twice

Question **29**

Correct

Mark 0.25 out of 0.25

What is the OS Kernel?

- ☒ a. The code that controls hardware, abstracts access to hardware resources using system calls, creates an environment for processes to be created and run ✔ correct
- ☐ b. The set of tools like compiler, linker, loader, terminal, shell, etc.
- ☐ c. Only the system programs like compiler, linker, loader, etc.
- ☐ d. Everything that I see on my screen

The correct answer is: The code that controls hardware, abstracts access to hardware resources using system calls, creates an environment for processes to be created and run

Question **30**

Correct

Mark 0.50 out of 0.50

Which of the following is/are not saved during context switch?

- ☐ a. Program Counter
- ☐ b. General Purpose Registers
- ☒ c. Bus ✔
- ☐ d. Stack Pointer
- ☐ e. MMU related registers/information
- ☒ f. Cache ✔
- ☒ g. TLB ✔

Your answer is correct.

The correct answers are: TLB, Cache, Bus

Question 31

Partially correct

Mark 0.10 out of 0.25

Select the order in which the various stages of a compiler execute.

Linking	3	✗
Syntactical Analysis	2	✓
Pre-processing	1	✓
Intermediate code generation	does not exist	✗
Loading	4	✗

Your answer is partially correct.

You have correctly selected 2.

The correct answer is: Linking → 4, Syntactical Analysis → 2, Pre-processing → 1, Intermediate code generation → 3, Loading → does not exist

Question 32

Partially correct

Mark 0.08 out of 0.50

Order the sequence of events, in scheduling process P1 after process P0

context of P0 is saved in P0's PCB	2	✗
context of P1 is loaded from P1's PCB	3	✗
Process P1 is running	5	✗
timer interrupt occurs	6	✗
Process P0 is running	1	✓
Control is passed to P1	4	✗

Your answer is partially correct.

You have correctly selected 1.

The correct answer is: context of P0 is saved in P0's PCB → 3, context of P1 is loaded from P1's PCB → 4, Process P1 is running → 6, timer interrupt occurs → 2, Process P0 is running → 1, Control is passed to P1 → 5

Question **33**

Not answered

Marked out of 1.00

Select the correct statements about interrupt handling in xv6 code

- ☐ a. On any interrupt/syscall/exception the control first jumps in vectors.S
- ☐ b. The trapframe pointer in struct proc, points to a location on user stack
- ☐ c. Each entry in IDT essentially gives the values of CS and EIP to be used in handling that interrupt
- ☐ d. xv6 uses the 64th entry in IDT for system calls
- ☐ e. The CS and EIP are changed only after pushing user code's SS,ESP on stack
- ☐ f. The trapframe pointer in struct proc, points to a location on kernel stack
- ☐ g. The function trap() is the called only in case of hardware interrupt
- ☐ h. The CS and EIP are changed only immediately on a hardware interrupt
- ☐ i. All the 256 entries in the IDT are filled
- ☐ j. On any interrupt/syscall/exception the control first jumps in trapasm.S
- ☐ k. The function trap() is the called irrespective of hardware interrupt/system-call/exception
- ☐ l. xv6 uses the 0x64th entry in IDT for system calls
- ☐ m. Before going to alltraps, the kernel stack contains upto 5 entries.

Your answer is incorrect.

The correct answers are: All the 256 entries in the IDT are filled, Each entry in IDT essentially gives the values of CS and EIP to be used in handling that interrupt, xv6 uses the 64th entry in IDT for system calls, On any interrupt/syscall/exception the control first jumps in vectors.S, Before going to alltraps, the kernel stack contains upto 5 entries., The trapframe pointer in struct proc, points to a location on kernel stack, The function trap() is the called irrespective of hardware interrupt/system-call/exception, The CS and EIP are changed only after pushing user code's SS,ESP on stack

◀ (Assignment) [Change free list management in xv6](#)

Jump to...

[Dashboard](#) / [My courses](#) / [Computer Engineering & IT](#) / [CEIT-Even-sem-20-21](#) / [OS-Even-sem-2020-21](#) / 14 March - 20 March
/ [Quiz - 2 \(18 March\)](#)

Started on Thursday, 18 March 2021, 2:46 PM

State Finished

Completed on Thursday, 18 March 2021, 3:50 PM

Time taken 1 hour 4 mins

Grade 10.36 out of 20.00 (52%)

Question 1

Partially correct

Mark 0.57 out of 1.00

Mark True, the actions done as part of code of switch() in switch.S, in xv6

True	False		
<input checked="" type="radio"/>	<input type="radio"/>	Restore new callee saved registers from kernel stack of new context	✓
<input checked="" type="radio"/>	<input type="radio"/>	Save old callee saved registers on kernel stack of old context	✓
<input type="radio"/>	<input checked="" type="radio"/>	Save old callee saved registers on user stack of old context	✓
<input type="radio"/>	<input checked="" type="radio"/>	Switch from old process context to new process context	✗
<input checked="" type="radio"/>	<input type="radio"/>	Switch from one stack (old) to another(new)	✗
<input type="radio"/>	<input checked="" type="radio"/>	Restore new callee saved registers from user stack of new context	✓
<input type="radio"/>	<input checked="" type="radio"/>	Jump to code in new context	✗

Restore new callee saved registers from kernel stack of new context: True

Save old callee saved registers on kernel stack of old context: True

Save old callee saved registers on user stack of old context: False

Switch from old process context to new process context: False

Switch from one stack (old) to another(new): True

Restore new callee saved registers from user stack of new context: False

Jump to code in new context: False

Question 2

Partially correct

Mark 0.17 out of 0.50

For each function/code-point, select the status of segmentation setup in xv6

bootmain()	gdt setup with 3 entries, right from first line of code of bootloader	✗
kvmalloc() in main()	gdt setup with 5 entries (0 to 4) on one processor	✗
after startothers() in main()	gdt setup with 5 entries (0 to 4) on all processors	✓
after seginit() in main()	gdt setup with 5 entries (0 to 4) on all processors	✗
bootasm.S	gdt setup with 3 entries, right from first line of code of bootloader	✗
entry.S	gdt setup with 3 entries, at start32 symbol of bootasm.S	✓

Your answer is partially correct.

You have correctly selected 2.

The correct answer is: bootmain() → gdt setup with 3 entries, at start32 symbol of bootasm.S, kvmalloc() in main() → gdt setup with 3 entries, at start32 symbol of bootasm.S, after startothers() in main() → gdt setup with 5 entries (0 to 4) on all processors, after seginit() in main() → gdt setup with 5 entries (0 to 4) on one processor, bootasm.S → gdt setup with 3 entries, at start32 symbol of bootasm.S, entry.S → gdt setup with 3 entries, at start32 symbol of bootasm.S

Question 3

Partially correct

Mark 0.38 out of 1.00

Compare paging with demand paging and select the correct statements.

Select one or more:

- ☒ a. The meaning of valid-invalid bit in page table is different in paging and demand-paging. ✓
- ☒ b. Demand paging requires additional hardware support, compared to paging. ✓
- ☐ c. Paging requires some hardware support in CPU
- ☒ d. With paging, it's possible to have user programs bigger than physical memory. ✗
- ☒ e. Both demand paging and paging support shared memory pages. ✓
- ☐ f. Demand paging always increases effective memory access time.
- ☒ g. With demand paging, it's possible to have user programs bigger than physical memory. ✓
- ☒ h. Calculations of number of bits for page number and offset are same in paging and demand paging. ✓
- ☐ i. TLB hit ration has zero impact in effective memory access time in demand paging.
- ☐ j. Paging requires NO hardware support in CPU

Your answer is partially correct.

You have correctly selected 5.

The correct answers are: Demand paging requires additional hardware support, compared to paging., Both demand paging and paging support shared memory pages., With demand paging, it's possible to have user programs bigger than physical memory., Demand paging always increases effective memory access time., Paging requires some hardware support in CPU, Calculations of number of bits for page number and offset are same in paging and demand paging., The meaning of valid-invalid bit in page table is different in paging and demand-paging.

Question 4

Partially correct

Mark 0.44 out of 0.50

Suppose a processor supports base(relocation register) + limit scheme of MMU.

Assuming this, mark the statements as True/False

True	False		
<input checked="" type="radio"/>	<input type="radio"/>	The OS may terminate the process while handling the interrupt of memory violation	✓
<input checked="" type="radio"/>	<input type="radio"/>	The hardware detects any memory access beyond the limit value and raises an interrupt	✓
<input type="radio"/>	<input checked="" type="radio"/>	The hardware may terminate the process while handling the interrupt of memory violation	✗
<input checked="" type="radio"/>	<input type="radio"/>	The OS sets up the relocation and limit registers when the process is scheduled	✓
<input checked="" type="radio"/>	<input type="radio"/>	The compiler generates machine code assuming continuous memory address space for process, and calculating appropriate sizes for code, and data;	✓
<input type="radio"/>	<input checked="" type="radio"/>	The process sets up it's own relocation and limit registers when the process is scheduled	✓
<input type="radio"/>	<input checked="" type="radio"/>	The OS detects any memory access beyond the limit value and raises an interrupt	✓
<input type="radio"/>	<input checked="" type="radio"/>	The compiler generates machine code assuming appropriately sized segments for code, data and stack.	✓

The OS may terminate the process while handling the interrupt of memory violation: True

The hardware detects any memory access beyond the limit value and raises an interrupt: True

The hardware may terminate the process while handling the interrupt of memory violation: False

The OS sets up the relocation and limit registers when the process is scheduled: True

The compiler generates machine code assuming continuous memory address space for process, and calculating appropriate sizes for code, and data;: True

The process sets up it's own relocation and limit registers when the process is scheduled: False

The OS detects any memory access beyond the limit value and raises an interrupt: False

The compiler generates machine code assuming appropriately sized segments for code, data and stack.: False

Question 5

Correct

Mark 0.50 out of 0.50

Consider the following list of free chunks, in continuous memory management:

10k, 25k, 12k, 7k, 9k, 13k

Suppose there is a request for chunk of size 9k, then the free chunk selected under each of the following schemes will be

Best fit:

9k



First fit:

10k



Worst fit:

25k



Question 6

Partially correct

Mark 0.50 out of 1.00

Select all the correct statements about MMU and it's functionality

Select one or more:

- ☐ a. MMU is a separate chip outside the processor
- ☒ b. MMU is inside the processor
- ☐ c. Logical to physical address translations in MMU are done with specific machine instructions
- ☒ d. The operating system interacts with MMU for every single address translation
- ☒ e. Illegal memory access is detected in hardware by MMU and a trap is raised
- ☐ f. The Operating system sets up relevant CPU registers to enable proper MMU translations
- ☒ g. Logical to physical address translations in MMU are done in hardware, automatically
- ☐ h. Illegal memory access is detected by operating system



Your answer is partially correct.

You have correctly selected 3.

The correct answers are: MMU is inside the processor, Logical to physical address translations in MMU are done in hardware, automatically, The Operating system sets up relevant CPU registers to enable proper MMU translations, Illegal memory access is detected in hardware by MMU and a trap is raised

Question 7

Incorrect

Mark 0.00 out of 0.50

Assuming a 8- KB page size, what is the page numbers for the address 874815 reference in decimal :
(give answer also in decimal)

Answer: 2186 ✖

The correct answer is: 107

Question 8

Incorrect

Mark 0.00 out of 0.25

Select the compiler's view of the process's address space, for each of the following MMU schemes:
(Assume that each scheme,e.g. paging/segmentation/etc is effectively utilised)

Segmentation, then paging	Many continuous chunks each of page size	✖
Relocation + Limit	Many continuous chunks of same size	✖
Segmentation	one continuous chunk	✖
Paging	many continuous chunks of variable size	✖

Your answer is incorrect.

The correct answer is: Segmentation, then paging → many continuous chunks of variable size, Relocation + Limit → one continuous chunk, Segmentation → many continuous chunks of variable size, Paging → one continuous chunk

Question 9

Incorrect

Mark 0.00 out of 0.50

Suppose the memory access time is 180ns and TLB hit ratio is 0.3, then effective memory access time is (in nanoseconds);

Answer: 192 ✖

The correct answer is: 306.00

Question 10

Correct

Mark 0.50 out of 0.50

In xv6, The struct context is given as

```
struct context {
    uint edi;
    uint esi;
    uint ebx;
    uint ebp;
    uint eip;
};
```

Select all the reasons that explain why only these 5 registers are included in the struct context.

- ☒ a. The segment registers are same across all contexts, hence they need not be saved ✓
- ☒ b. esp is not saved in context, because context{} is on stack and it's address is always argument to switch() ✓
- ☐ c. xv6 tries to minimize the size of context to save memory space
- ☐ d. esp is not saved in context, because it's not part of the context
- ☒ e. eax, ecx, edx are caller save, hence no need to save ✓

Your answer is correct.

The correct answers are: The segment registers are same across all contexts, hence they need not be saved, eax, ecx, edx are caller save, hence no need to save, esp is not saved in context, because context{} is on stack and it's address is always argument to switch()

Question 11

Partially correct

Mark 0.83 out of 1.50

Arrange the following events in order, in page fault handling:

Disk interrupt wakes up the process	7	✓
The reference bit is found to be invalid by MMU	1	✓
OS makes available an empty frame	6	✗
Restart the instruction that caused the page fault	9	✓
A hardware interrupt is issued	3	✗
OS schedules a disk read for the page (from backing store)	5	✓
Process is kept in wait state	4	✗
Page tables are updated for the process	8	✓
Operating system decides that the page was not in memory	2	✗

Your answer is partially correct.

You have correctly selected 5.

The correct answer is: Disk interrupt wakes up the process → 7, The reference bit is found to be invalid by MMU → 1, OS makes available an empty frame → 4, Restart the instruction that caused the page fault → 9, A hardware interrupt is issued → 2, OS schedules a disk read for the page (from backing store) → 5, Process is kept in wait state → 6, Page tables are updated for the process → 8, Operating system decides that the page was not in memory → 3

Question **12**

Incorrect

Mark 0.00 out of 0.50

Suppose a kernel uses a buddy allocator. The smallest chunk that can be allocated is of size 32 bytes. One bit is used to track each such chunk, where 1 means allocated and 0 means free. The chunk looks like this as of now:

00001010

Now, there is a request for a chunk of 70 bytes.

After this allocation, the bitmap, indicating the status of the buddy allocator will be

Answer: 11101010



The correct answer is: 11111010

Question **13**

Incorrect

Mark 0.00 out of 0.25

The complete range of virtual addresses (after main() in main.c is over), from which the free pages used by kalloc() and kfree() is derived, are:

- ☐ a. end, 4MB
- ☐ b. P2V(end), P2V(PHYSTOP)
- ☐ c. end, P2V(4MB + PHYSTOP)
- ☒ d. P2V(end), PHYSTOP
- ☐ e. end, (4MB + PHYSTOP)
- ☐ f. end, PHYSTOP
- ☐ g. end, P2V(PHYSTOP)



Your answer is incorrect.

The correct answer is: end, P2V(PHYSTOP)

Question 14

Partially correct

Mark 0.33 out of 0.50

Match the pair

Hashed page table	Linear search on collision done by OS (e.g. SPARC Solaris) typically	✓
Inverted Page table	Linear/Parallel search using frame number in page table	✗
Hierarchical Paging	More memory access time per hierarchy	✓

Your answer is partially correct.

You have correctly selected 2.

The correct answer is: Hashed page table → Linear search on collision done by OS (e.g. SPARC Solaris) typically, Inverted Page table → Linear/Parallel search using page number in page table, Hierarchical Paging → More memory access time per hierarchy

Question 15

Partially correct

Mark 0.29 out of 0.50

After virtual memory is implemented

(select T/F for each of the following) One Program's size can be larger than physical memory size

True	False		
<input checked="" type="radio"/>	<input type="radio"/>	Code need not be completely in memory	✓
<input checked="" type="radio"/>	<input type="radio"/>	Cumulative size of all programs can be larger than physical memory size	✓
<input type="radio"/>	<input checked="" type="radio"/>	Virtual access to memory is granted	✗
<input checked="" type="radio"/>	<input type="radio"/>	Logical address space could be larger than physical address space	✓
<input type="radio"/>	<input checked="" type="radio"/>	Virtual addresses are available	✗
<input type="radio"/>	<input checked="" type="radio"/>	Relatively less I/O may be possible during process execution	✗
<input checked="" type="radio"/>	<input type="radio"/>	One Program's size can be larger than physical memory size	✓

Code need not be completely in memory: True

Cumulative size of all programs can be larger than physical memory size: True

Virtual access to memory is granted: False

Logical address space could be larger than physical address space: True

Virtual addresses are available: False

Relatively less I/O may be possible during process execution: True

One Program's size can be larger than physical memory size: True

Question **16**

Partially correct

Mark 0.64 out of 1.00

W.r.t. Memory management in xv6,

xv6 uses physical memory upto 224 MB only Mark statements True or False

True	False		
<input checked="" type="radio"/>	<input type="radio"/>	The switchkvm() call in scheduler() is invoked after control comes to it from sched(), thus demanding execution in kernel's context	✓
<input checked="" type="radio"/>	<input type="radio"/>	The stack allocated in entry.S is used as stack for scheduler's context for first processor	✓
<input checked="" type="radio"/>	<input type="radio"/>	The switchkvm() call in scheduler() changes CR3 to use page directory kpgdir	✓
<input type="radio"/>	<input checked="" type="radio"/>	The free page-frame are created out of nearly 222 MB	✗
<input checked="" type="radio"/>	<input type="radio"/>	The kernel code and data take up less than 2 MB space	✓
<input type="radio"/>	<input checked="" type="radio"/>	The switchkvm() call in scheduler() changes CR3 to use page directory of new process	✗
<input type="radio"/>	<input checked="" type="radio"/>	The switchkvm() call in scheduler() is invoked after control comes to it from swtch() scheduler(), thus demanding execution in new process's context	✓
<input checked="" type="radio"/>	<input type="radio"/>	PHYSTOP can be increased to some extent, simply by editing memlayout.h	✓
<input type="radio"/>	<input checked="" type="radio"/>	xv6 uses physical memory upto 224 MB only	✗
<input checked="" type="radio"/>	<input type="radio"/>	The process's address space gets mapped on frames, obtained from ~2MB:224MB range	✓
<input type="radio"/>	<input checked="" type="radio"/>	The kernel's page table given by kpgdir variable is used as stack for scheduler's context	✗

The switchkvm() call in scheduler() is invoked after control comes to it from sched(), thus demanding execution in kernel's context: True

The stack allocated in entry.S is used as stack for scheduler's context for first processor: True

The switchkvm() call in scheduler() changes CR3 to use page directory kpgdir: True

The free page-frame are created out of nearly 222 MB: True

The kernel code and data take up less than 2 MB space: True

The switchkvm() call in scheduler() changes CR3 to use page directory of new process: False

The switchkvm() call in scheduler() is invoked after control comes to it from swtch() scheduler(), thus demanding execution in new process's context: False

PHYSTOP can be increased to some extent, simply by editing memlayout.h: True

xv6 uses physical memory upto 224 MB only: True

The process's address space gets mapped on frames, obtained from ~2MB:224MB range: True

The kernel's page table given by kpgdir variable is used as stack for scheduler's context: False

Question **17**


Incorrect

Mark 0.00 out of 1.50

Consider the reference string

6 4 2 0 1 2 6 9 2 0 5

If the number of page frames is 3, then total number of page faults (including initial), using LRU replacement is:

Answer: 

#6# 6,4# 6,4,2 # 0,4,2#0,1,2#6,1,2#6,9,2#0,9,2#0,5,2

The correct answer is: 9

Question 18

Partially correct

Mark 0.31 out of 0.50

Consider the image given below, which explains how paging works.

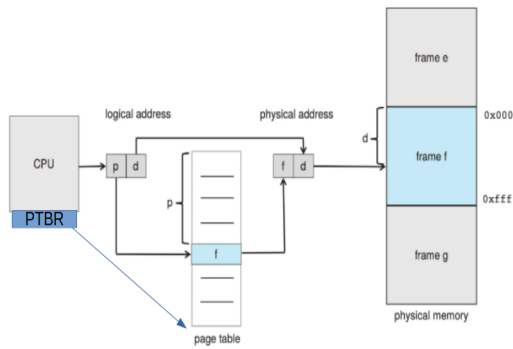


Figure 9.8 Paging hardware.

Mention whether each statement is True or False, with respect to this image.

True	False		
<input checked="" type="radio"/>	<input type="radio"/>	The PTBR is present in the CPU as a register	✓
<input type="radio"/>	<input checked="" type="radio"/>	The page table is indexed using frame number	✓
<input checked="" type="radio"/>	<input type="radio"/>	The page table is indexed using page number	✗
<input checked="" type="radio"/>	<input type="radio"/>	The locating of the page table using PTBR also involves paging translation	✗
<input type="radio"/>	<input checked="" type="radio"/>	Size of page table is always determined by the size of RAM	✓
<input checked="" type="radio"/>	<input type="radio"/>	The page table is itself present in Physical memory	✓
<input checked="" type="radio"/>	<input type="radio"/>	Maximum Size of page table is determined by number of bits used for page number	✗
<input checked="" type="radio"/>	<input type="radio"/>	The physical address may not be of the same size (in bits) as the logical address	✓

The PTBR is present in the CPU as a register: True

The page table is indexed using frame number: False

The page table is indexed using page number: True

The locating of the page table using PTBR also involves paging translation: False

Size of page table is always determined by the size of RAM: False

The page table is itself present in Physical memory: True

Maximum Size of page table is determined by number of bits used for page number: True

The physical address may not be of the same size (in bits) as the logical address: True

Question **19**

Correct

Mark 2.00 out of 2.00

Given below is shared memory code with two processes sharing a memory segment.

The first process sends a user input string to second process. The second capitalizes the string. Then the first process prints the capitalized version.

Fill in the blanks to complete the code.

// First process

#define SHMSZ 27

int main()

{

char c;

int shmid;

key_t key;

char *shm, *s, string[128];

key = 5679;

if ((shmid =

shmget

✓ (key, SHMSZ, IPC_CREAT | 0666)) < 0) {

perror("shmget");

exit(1);

}

if ((shm =

shmat

✓ (shmid, NULL, 0)) == (char *) -1) {

perror("shmat");

exit(1);

}

s = shm;

*s = '\$';

scanf("%s", string);

strcpy(s + 1, string);

*s = '

@

✓ '; //note the quotes

while(*s != ')

\$

✓ ')

sleep(1);

printf("%s\n", s + 1);

exit(0);

}

//Second process

#define SHMSZ 27

int main()

{

int shmid;

key_t key;

char *shm, *s;

int i;

char string[128];

key =

5679

```

✓ ;
if ((shm = shmget(key, SHMSZ, 0666)) < 0) {
    perror("shmget");
    exit(1);
}
if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
    perror("shmat");
    exit(1);
}
s =

```

shm

```

✓ ;
while(*s != '@')
    sleep(1);
for(i = 0; i < strlen(s + 1); i++)
    s[i + 1] = toupper(s[i + 1]);
*s = '$';
exit(0);
}

```

Question 20

Partially correct

Mark 0.25 out of 0.50

Map the functionality/use with function/variable in xv6 code.

return a free page, if available; 0, otherwise

kinit1()

✗

Create page table entries for a given range of virtual and physical addresses; including page directory entries if needed

mappages()

✓

Array listing the kernel memory mappings, to be used by setupkvm()

kmap[]

✓

Setup kernel part of a page table, mapping kernel code, data, read-only data, I/O space, devices

kvmalloc()

✗

Return address of page table entry in a given page directory, for a given virtual address; creates page table if necessary

walkpgdir()

✓

Setup kernel part of a page table, and switch to that page table

setupkvm()

✗

Your answer is partially correct.

You have correctly selected 3.

The correct answer is: return a free page, if available; 0, otherwise → kalloc(), Create page table entries for a given range of virtual and physical addresses; including page directory entries if needed → mappages(), Array listing the kernel memory mappings, to be used by setupkvm() → kmap[], Setup kernel part of a page table, mapping kernel code, data, read-only data, I/O space, devices → setupkvm(), Return address of page table entry in a given page directory, for a given virtual address; creates page table if necessary → walkpgdir(), Setup kernel part of a page table, and switch to that page table → kvmalloc()

Question **21**

Partially correct

Mark 1.53 out of 2.50

Order events in xv6 timer interrupt code

(Transition from process P1 to P2's code.)

P2 is selected and marked RUNNING	12	✓
Change of stack from user stack to kernel stack of P1	3	✓
Timer interrupt occurs	2	✓
alltraps() will call iret	17	✗
change to context of P2, P2's kernel stack in use now	13	✓
P2's trap() will return to alltraps	16	✗
jump in vector.S	4	✓
P2 will return from sched() in yield()	14	✗
yield() is called	8	✓
trap() is called	7	✓
Process P2 is executing	18	✗
P1 is marked as RUNNABLE	9	✓
P2's yield() will return in trap()	15	✗
Process P1 is executing	1	✓
sched() is called,	11	✗
change to context of the scheduler, scheduler's stack in use now	10	✗
jump to alltraps	5	✓
Trapframe is built on kernel stack of P1	6	✓

Your answer is partially correct.

You have correctly selected 11.

The correct answer is: P2 is selected and marked RUNNING → 12, Change of stack from user stack to kernel stack of P1 → 3, Timer interrupt occurs → 2, alltraps() will call iret → 18, change to context of P2, P2's kernel stack in use now → 13, P2's trap() will return to alltraps → 17, jump in vector.S → 4, P2 will return from sched() in yield() → 15, yield() is called → 8, trap() is called → 7, Process P2 is executing → 14, P1 is marked as RUNNABLE → 9, P2's yield() will return in trap() → 16, Process P1 is executing → 1, sched() is called, → 10, change to context of the scheduler, scheduler's stack in use now → 11, jump to alltraps → 5, Trapframe is built on kernel stack of P1 → 6

Question **22**

Incorrect

Mark 0.00 out of 1.00

Given that the memory access time is 200 ns, probability of a page fault is 0.7 and page fault handling time is 8 ms,
The effective memory access time in nanoseconds is:

Answer: ❌

The correct answer is: 5600060.00

Question **23**

Correct

Mark 0.25 out of 0.25

Select the state that is not possible after the given state, for a process:

New: ✔️

Ready : ✔️

Running: : ✔️

Waiting: ✔️

Question **24**

Partially correct

Mark 0.63 out of 1.00

Select the correct statements about sched() and scheduler() in xv6 code

- ☒ a. scheduler() switches to the selected process's context ✔️
- ☒ b. When either sched() or scheduler() is called, it does not return immediately to caller ✔️
- ☐ c. After call to switch() in sched(), the control moves to code in scheduler()
- ☒ d. Each call to sched() or scheduler() involves change of one stack inside switch() ✔️
- ☐ e. After call to switch() in scheduler(), the control moves to code in sched()
- ☒ f. When either sched() or scheduler() is called, it results in a context switch ✔️
- ☒ g. sched() switches to the scheduler's context ✔️
- ☐ h. sched() and scheduler() are co-routines

Your answer is partially correct.

You have correctly selected 5.

The correct answers are: sched() and scheduler() are co-routines, When either sched() or scheduler() is called, it does not return immediately to caller, When either sched() or scheduler() is called, it results in a context switch, sched() switches to the scheduler's context, scheduler() switches to the selected process's context, After call to switch() in scheduler(), the control moves to code in sched(), After call to switch() in sched(), the control moves to code in scheduler(), Each call to sched() or scheduler() involves change of one stack inside switch()

Question **25**

Correct

Mark 0.25 out of 0.25

The data structure used in `kalloc()` and `kfree()` in `xv6` is

- ☐ a. Doubly linked circular list
- ☐ b. Singly linked circular list
- ☐ c. Double linked NULL terminated list
- ☒ d. Singly linked NULL terminated list



Your answer is correct.

The correct answer is: Singly linked NULL terminated list

[◀ \(Assignment\) lseek system call in xv6](#)

Jump to...