readable medium, such as a CD-ROM, a volatile memory, a non-volatile memory, ROM, RAM, or any other suitable storage device.

The processing capability of the system **100** may be distributed among multiple entities, such as among multiple processors and memories, optionally including multiple distributed processing systems. Parameters, databases, and other data structures may be separately stored and managed, may be incorporated into a single memory or database, may be logically and physically organized in many different ways, and may implemented with different types of data structures such as linked lists, hash tables, or implicit storage mechanisms. Logic, such as programs or circuitry, may be combined or split among multiple programs, distributed across several memories and processors, and may be implemented in a library, such as a shared library (for example, a dynamic link library (DLL)).

All of the discussion, regardless of the particular implementation described, is exemplary in nature, rather than limiting. For example, although selected aspects, features, or components of the implementations are depicted as being stored in memories, all or part of the system **100** or systems may be stored on, distributed across, or read from other computer readable storage media, for example, secondary storage devices such as hard disks, flash memory drives, floppy disks, and CD-ROMs. Moreover, the various modules and screen display functionality is but one example of such functionality and any other configurations encompassing similar functionality are possible.

The respective logic, software or instructions for implementing the processes, methods and/or techniques discussed above may be provided on computer readable storage media. The functions, acts or tasks illustrated in the figures or described herein may be executed in response to one or more sets of logic or instructions stored in or on computer readable media. The functions, acts or tasks are independent of the particular type of instructions set, storage media, processor or processing strategy and may be performed by software, hardware, integrated circuits, firmware, micro code and the like, operating alone or in combination. Likewise, processing strategies may include multiprocessing, multitasking, parallel processing and the like. In one embodiment, the instructions are stored on a removable media device for reading by local or remote systems. In other embodiments, the logic or instructions are stored in a remote location for transfer through a computer network or over telephone lines. In yet other embodiments, the logic or instructions are stored within a given computer, central processing unit ("CPU"), graphics processing unit ("GPU"), or system.

Furthermore, although specific components are described above, methods, systems, and articles of manufacture described herein may include additional, fewer, or different components. For example, a processor may be implemented as a microprocessor, microcontroller, application specific integrated circuit (ASIC), discrete logic, or a combination of other type of circuits or logic. Similarly, memories may be DRAM, SRAM, Flash or any other type of memory. Flags, data, databases, tables, entities, and other data structures may be separately stored and managed, may be incorporated into a single memory or database, may be distributed, or may be logically and physically organized in many different ways. The components may operate independently or be part of a same program or apparatus. The components may be resident on separate hardware, such as separate removable circuit boards, or share common hardware, such as a same memory and processor for implementing instructions from the memory. Programs may be parts of a single program, separate programs, or distributed across several memories and processors.

Although specific steps of methods are illustrated in flow diagrams, additional, fewer, or different steps may be included in the illustrated methods. In addition, the steps may be performed in an order different than illustrated.

A second action may be said to be "in response to" a first action independent of whether the second action results directly or indirectly from the first action. The second action may occur at a substantially later time than the first action and still be in response to the first action. Similarly, the second action may be said to be in response to the first action even if intervening actions take place between the first action and the second action, and even if one or more of the intervening actions directly cause the second action to be performed. For example, a second action may be in response to a first action if the first action sets a flag and a third action later initiates the second action whenever the flag is set.

To clarify the use of and to hereby provide notice to the public, the phrases "at least one of <A>, <B>, . . . and <N>" or "at least one of <A>, <B>, <N>, or combinations thereof" or "<A>, <B>, . . . and/or <N>" are defined by the Applicant in the broadest sense, superseding any other implied definitions hereinbefore or hereinafter unless expressly asserted by the Applicant to the contrary, to mean one or more elements selected from the group comprising A, B, . . . and N. In other words, the phrases mean any combination of one or more of the elements A, B, . . . or N including any one element alone or the one element in combination with one or more of the other elements which may also include, in combination, additional elements not listed.

While various embodiments have been described, it will be apparent to those of ordinary skill in the art that many more embodiments and implementations are possible. Accordingly, the embodiments described herein are examples, not the only possible embodiments and implementations.

What is claimed is:

1. A system comprising:

a memory comprising a mapping of a first portion of a memory-mapped file to a virtual address for a first process, wherein the memory-mapped file comprises virtual memory backed by a file; and

a processor configured to:

map a second portion of the memory-mapped file to the virtual address for a second process based on the second process being forked from the first process, wherein the first and second portions of the memory-mapped file are backed by the file; and

write data from the first and second portions of the memory-mapped file to corresponding first and second portions of the file that backs the memory-mapped file.

2. The system of claim **1**, wherein the processor is configured to create a copy of data associated with the first portion of the memory-mapped file in response to a write to the virtual address by the first process or the second process.

3. The system of claim **1**, wherein the processor is configured to create a copy of data associated with the first portion of the memory-mapped file when the second process is forked from the first process.

4. The system of claim **1**, wherein the processor is configured to create a copy of data associated with the first portion of the memory-mapped file in response to a memory access operation on the virtual address by the first process or the second process.

5. The system of claim **1**, wherein the file is a pseudo file.

6. A method for providing fork-safe memory allocation from memory-mapped files, the method comprising:

allocating a first portion of a memory-mapped file to a virtual address for a first process;

allocating a second portion of the memory-mapped file to the virtual address for a second process based on the second process being a child of the first process, wherein the first portion of the memory-mapped file and the second portion of the memory-mapped file are mapped to a first offset and a second offset, respectively, of a file that backs the memory-mapped file; and

writing contents of the first and second portions of the memory-mapped file to corresponding first and second portions of the file that backs the memory-mapped file.

7. The method of claim 6, further comprising selecting the second offset of the file for the second portion of the memory-mapped file when the virtual address is accessed and/or written by the child of the first process.

8. The method of claim 6, wherein the file includes a first file and a second file, and wherein the first portion of the memory-mapped file is mapped to the first offset of the first file and the second portion is mapped to the second offset of the second file.

9. The method of claim 6, further comprising encrypting file data stored in a primary memory in response to a write to the virtual address before writing the encrypted file data from the primary memory to the file that backs the memory-mapped file.

10. The method of claim 6 further comprising mapping the memory-mapped file to a pseudo file, and translating reads and writes to the pseudo file into reads and writes over a network to a memory appliance instead of accessing data in a file system.

11. The method of claim 10, wherein the translating comprises reading from and/or writing to memory of the memory appliance via a memory access protocol independently of a central processing unit of the memory appliance.

12. The method of claim 6, wherein the file includes an interface that provides access over a network to a corresponding area of primary memory of a memory appliance.

13. An apparatus comprising:

a processor; and

a memory, the memory comprising:

a first virtual address space for a first process executed by the processor;

a second virtual address space for a second process executed by the processor, wherein a virtual address in the first virtual address space is also in the second virtual address space;

at least a subset of a memory-mapped file, the memory-mapped file comprising a first portion and a second portion that include virtual memory backed by a file; and

a memory allocation module configured to map the virtual address in the first virtual address space for the first process to the first portion of the memory-mapped file and to map the virtual address in the second virtual address space for the second process to the second portion of the memory-mapped file based on the second process being a child of the first process, wherein the memory allocation module is further configured to write contents of the first and second portions of the memory-mapped file to corresponding first and second portions of the file that backs the memory-mapped file.

14. The apparatus of claim 13, wherein the memory allocation module is further configured to encrypt data associated with the first portion or the second portion of the memory-mapped file before or as the data is written to the memory-mapped file, wherein the memory-mapped file includes virtual memory assigned to a pseudo file.

15. The apparatus of claim 14 wherein the memory allocation module is further configured to read the encrypted data from the memory-mapped file and decrypt the encrypted data after or as the encrypted data is read from the memory-mapped file.

16. The apparatus of claim 13, wherein the memory allocation module comprises executable instructions included in an operating system.

17. The apparatus of claim 13, wherein the memory allocation module comprises executable instructions included in a kernel or kernel module.

18. The apparatus of claim 13, wherein the memory allocation module comprises executable instructions included in a user library.

19. The apparatus of claim 13, wherein the memory-mapped file is mapped to an interface configured to provide access over a network to a corresponding area of primary memory of a memory appliance.

20. The apparatus of claim 19, wherein access over the network to the corresponding area of primary memory of the memory appliance is via a Remote Direct Memory Access (RDMA) protocol and is independent of a central processing unit of the memory appliance.

21. The apparatus of claim 13, wherein the first process and/or the second process are included in a container, a jail, and/or a zone.

22. An apparatus comprising:

a processor; and

a memory, the memory comprising:

a first virtual address space for a first process executed by the processor;

a second virtual address space for a second process executed by the processor, wherein a virtual address in the first virtual address space is also in the second virtual address space;

at least a subset of a memory-mapped file, the memory-mapped file comprising a first portion and a second portion that include virtual memory backed by a file; and

a memory allocation module configured to map the virtual address in the first virtual address space for the first process to the first portion of the memory-mapped file and to map the virtual address in the second virtual address space for the second process to the second portion of the memory-mapped file based on the second process being a child of the first process,

wherein the memory allocation module is further configured to encrypt data associated with the first portion or the second portion of the memory-mapped file before or as the data is written to the memory-mapped file, wherein the memory-mapped file includes virtual memory assigned to a pseudo file, and

wherein the memory allocation module is further configured to read the encrypted data from the memory-mapped file and decrypt the encrypted data after or as the encrypted data is read from the memory-mapped file.

* * * * *