A proof of Doug Lea's memory manager

John Wickerson

Chapter 1

Glossary of macros, typedefs and minor routines

`MALLOC_ALIGNMENT`	=	8
`MAX_SIZE_T`	=	$FFFF\ FFFF_h$
`SIZE_T_SIZE`	=	4
`SIZE_T_BITSIZE`	=	32
`SIZE_T_ZERO`	=	0
`SIZE_T_ONE`	=	1
`SIZE_T_TWO`	=	2
`SIZE_T_FOUR`	=	4
`TWO_SIZE_T_SIZES`	=	8
`FOUR_SIZE_T_SIZES`	=	16
`SIX_SIZE_T_SIZES`	=	24
`HALF_MAX_SIZE_T`	=	$7FFF\ FFFF_h$
`CHUNK_ALIGN_MASK`	=	111_b
`mchunk`	=	`struct malloc_chunk`
`mchunkptr`	=	`mchunk*`
`sbinptr`	=	`mchunk*`
`bindex_t`	=	`unsigned int`
`binmap_t`	=	`unsigned int`
`flag_t`	=	`unsigned int`
`MCHUNK_SIZE`	=	16
`CHUNK_OVERHEAD`	=	4
`MIN_CHUNK_SIZE`	=	16
`chunk2mem(p)`	=	$\texttt{p} + 8$
`mem2chunk(mem)`	=	$\texttt{mem} - 8$
`MAX_REQUEST`	=	$2^{32} - 63$
`MIN_REQUEST`	=	11
`pad_request(req)`	=	$\lceil \texttt{req} + 4 \rceil_8$
`request2size(req)`	=	$\max\{16, \lceil \texttt{req} + 4 \rceil_8\}$

`PINUSE_BIT` $= 1_b$

`CINUSE_BIT` $= 10_b$

`FLAG4_BIT` $= 100_b$

`INUSE_BITS` $= 11_b$

`FLAG_BITS` $= 111_b$

`cinuse(p)` $= \mathit{flags}(\mathtt{p}) = \blacktriangledown\_$

`pinuse(p)` $= \mathit{flags}(\mathtt{p}) = \_\blacktriangle$

`is_inuse(p)` $= \mathit{flags}(\mathtt{p}) \in \{\blacktriangledown\_, \triangledown\vartriangle\}$

`is_mmapped(p)` $= \mathit{flags}(\mathtt{p}) = \triangledown\vartriangle$

`chunksize(p)` $= \mathit{size}(\mathtt{p})$

$\{\mathit{flags}(\mathtt{p}) = C\_\}$ `clear_pinuse(p)` $\{\mathit{flags}(\mathtt{p}) = C\vartriangle\}$

`chunk_plus_offset(p,s)` $= p + s$

`chunk_minus_offset(p,s)` $= p - s$

`next_chunk(p)` $= \mathit{next}(\mathtt{p})$

`prev_chunk(p)` $= \mathit{prev}(\mathtt{p})$

`next_pinuse(p)` $= \mathit{flags}(\mathit{next}(\mathtt{p})) = \_\blacktriangle$

`get_foot(p,s)` $= \mathit{prev\_foot}(\mathtt{p} + \mathtt{s})$

$\{\mathit{prev\_foot}(\mathtt{p} + \mathtt{s}) = \_\}$ `set_foot(p,s)` $\{\mathit{prev\_foot}(\mathtt{p} + \mathtt{s}) = \mathtt{s}\}$

$\left\{\begin{array}{l}\mathit{size}(\mathtt{p}) = \_ \wedge \mathit{flags}(\mathtt{p}) = \_\_ \\ \wedge\, \mathit{prev\_foot}(\mathtt{p} + \mathtt{s}) = \_\end{array}\right\}$ `set_size_and_pinuse_of_free_chunk(p,s)` $\left\{\begin{array}{l}\mathit{size}(\mathtt{p}) = \mathtt{s} \wedge \mathit{flags}(\mathtt{p}) = \triangledown\blacktriangle \\ \wedge\, \mathit{prev\_foot}(\mathit{next}(\mathtt{p})) = \mathtt{s}\end{array}\right\}$

$\left\{\begin{array}{l}\mathit{size}(\mathtt{p}) = \_ \wedge \mathit{flags}(\mathtt{p}) = \_\_ \\ \wedge\, \mathit{prev\_foot}(\mathtt{p} + \mathtt{s}) = \_ \\ \wedge\, \mathit{flags}(\mathtt{p} + \mathtt{s}) = \_\_\end{array}\right\}$ `set_free_with_pinuse(p,s,n)` $\left\{\begin{array}{l}\mathit{size}(\mathtt{p}) = \mathtt{s} \wedge \mathit{flags}(\mathtt{p}) = \triangledown\blacktriangle \\ \wedge\, \mathit{prev\_foot}(\mathit{next}(\mathtt{p})) = \mathtt{s} \\ \wedge\, \mathit{flags}(\mathit{next}(\mathtt{p})) = \_\vartriangle\end{array}\right\}$

`tchunk` = `malloc_tree_chunk`

`tchunkptr` = `tchunk*`

`tbinptr` = `tchunk*`

`leftmost_child(t)` $= \begin{cases} \mathit{child}_0(*\mathtt{t}) & \text{if } \mathit{child}_0(*\mathtt{t}) \neq 0 \\ \mathit{child}_1(*\mathtt{t}) & \text{otherwise} \end{cases}$

`NSMALLBINS` $= 32$

`NTREEBINS` $= 32$

`SMALLBIN_SHIFT` $= 3$

`SMALLBIN_WIDTH` $= 8$

`TREEBIN_SHIFT` $= 8$

`MIN_LARGE_SIZE` $= 256$

`MAX_SMALL_SIZE` $= 255$

`MAX_SMALL_REQUEST` $= 244$

`mstate` = `struct malloc_state`

`mparams` = `struct malloc_params`

`is_small(s)` $= \mathtt{s} < 256$

`small_index(s)` $= \lfloor \mathtt{s}/8 \rfloor$

`small_index2size(i)` $= 8 \times \mathtt{i}$

`MIN_SMALL_INDEX` $= 2$

$\left\{\mathtt{smallbins}[2\mathtt{i}+2] \mapsto C_1 * \mathtt{smallbins}[2\mathtt{i}+3] \mapsto C_2\right\}$ `x := smallbin_at(M,i)` $\left\{\mathtt{x.fd} \mapsto C_1 * \mathtt{x.bk} \mapsto C_2\right\}$

$$\mathtt{treebin\_at(M,i)} = \mathtt{treebins}[\mathtt{i}]$$

$\left\{\mathtt{I} = \_\right\}$ `compute_tree_index(S,I)` $\left\{\mathtt{I} = \begin{cases} 0 & \text{if } \mathtt{S} < 256 \\ 31 & \text{if } \mathtt{S} > 2^{24} \\ 2(\log_2 \lVert\mathtt{S}\rVert - 8) & \text{if } 0 \le \{\!\{\mathtt{S}\}\!\} < \frac{1}{2}\lVert\mathtt{S}\rVert \\ 2(\log_2 \lVert\mathtt{S}\rVert - 8) + 1 & \text{if } \frac{1}{2}\lVert\mathtt{S}\rVert \le \{\!\{\mathtt{S}\}\!\} < \lVert\mathtt{S}\rVert \end{cases}\right\}$

$$\mathtt{bin\_for\_tree\_index(i)} = \begin{cases} 31 & \text{if } \mathtt{i} = 31 \\ \lfloor \mathtt{i}/2 \rfloor + 6 & \text{otherwise} \end{cases}$$

$$\mathtt{leftshift\_for\_tree\_index(i)} = \begin{cases} 0 & \text{if } \mathtt{i} = 31 \\ 25 - \lfloor \mathtt{i}/2 \rfloor & \text{otherwise} \end{cases}$$

$$\mathtt{minsize\_for\_tree\_index(i)} = \begin{cases} 2 \ll (\lfloor \mathtt{i}/2 \rfloor + 7) & \text{if } \mathtt{i} \text{ even} \\ 3 \ll (\lfloor \mathtt{i}/2 \rfloor + 7) & \text{if } \mathtt{i} \text{ odd} \end{cases}$$

$$\mathtt{idx2bit(i)} = 1 \ll \mathtt{i}$$

$\left\{\mathtt{smallmap}[\mathtt{i}] = \_\right\}$ `mark_smallmap(M,i)` $\left\{\mathtt{smallmap}[\mathtt{i}] = 1\right\}$

$\left\{\mathtt{smallmap}[\mathtt{i}] = \_\right\}$ `clear_smallmap(M,i)` $\left\{\mathtt{smallmap}[\mathtt{i}] = 0\right\}$

$$\mathtt{smallmap\_is\_marked(M,i)} = \mathtt{smallmap}[\mathtt{i}] = 1$$

$\left\{\mathtt{treemap}[\mathtt{i}] = \_\right\}$ `mark_treemap(M,i)` $\left\{\mathtt{treemap}[\mathtt{i}] = 1\right\}$

$\left\{\mathtt{treemap}[\mathtt{i}] = \_\right\}$ `clear_treemap(M,i)` $\left\{\mathtt{treemap}[\mathtt{i}] = 0\right\}$

$$\mathtt{treemap\_is\_marked(M,i)} = \mathtt{treemap}[\mathtt{i}] = 1$$

$$\mathtt{least\_bit(x)} = \begin{cases} \mathbf{0}\,\overset{i}{1}\,\mathbf{0} & \text{if } \mathtt{x}_i = 1 \wedge \forall j < i.\, \mathtt{x}_j = 0\} \\ \mathbf{0} & \text{if } \mathtt{x}{=}0 \end{cases}$$

$$\mathtt{left\_bits(x)} = \begin{cases} \mathbf{1}\,\overset{i}{0}\,\mathbf{0} & \text{if } \mathtt{x}_i = 1 \wedge \forall j < i.\, \mathtt{x}_j = 0\} \\ \mathbf{0} & \text{if } \mathtt{x} = 0 \end{cases}$$

$$\mathtt{same\_or\_left\_bits(x)} = \begin{cases} \mathbf{1}\,\overset{i}{1}\,\mathbf{0} & \text{if } \mathtt{x}_i = 1 \wedge \forall j < i.\, \mathtt{x}_j = 0\} \\ \mathbf{0} & \text{if } \mathtt{x} = 0 \end{cases}$$

$\left\{\mathtt{I} = \_\right\}$ `compute_bit2idx(X,I)` $\left\{\mathtt{X} \neq 0 \Rightarrow \mathtt{I} = \log_2 \mathtt{X}\right\}$

$\left\{p\right\}$ `mark_inuse_foot(M,p,s)` $\left\{p\right\}$

$\left\{\begin{array}{l} size(\mathtt{p}) = \_ \wedge flags(\mathtt{p}) = \_P \\ \wedge\, flags(\mathtt{p}+\mathtt{s}) = C\_ \end{array}\right\}$ `set_inuse(M,p,s)` $\left\{\begin{array}{l} size(\mathtt{p}) = \mathtt{s} \wedge flags(\mathtt{p}) = \blacktriangledown P \\ \wedge\, flags(next(\mathtt{p})) = C\blacktriangle \end{array}\right\}$

$\left\{\begin{array}{l} size(\mathtt{p}) = \_ \wedge flags(\mathtt{p}) = \_\_ \\ \wedge\, flags(\mathtt{p}+\mathtt{s}) = C\_ \end{array}\right\}$ `set_inuse_and_pinuse(M,p,s)` $\left\{\begin{array}{l} size(\mathtt{p}) = \mathtt{s} \wedge flags(\mathtt{p}) = \blacktriangledown\blacktriangle \\ \wedge\, flags(next(\mathtt{p})) = C\blacktriangle \end{array}\right\}$

$\left\{\begin{array}{l} size(\mathtt{p}) = \_ \\ \wedge\, flags(\mathtt{p}) = \_\_ \end{array}\right\}$ `set_inuse_and_pinuse_of_inuse_chunk(M,p,s)` $\left\{\begin{array}{l} size(\mathtt{p}) = \mathtt{s} \\ \wedge\, flags(\mathtt{p}) = \blacktriangledown\blacktriangle \end{array}\right\}$

Chapter 2

Describing chunks

Define

$$
\begin{array}{rcl}
C_1 \curvearrowright C_2 & \Leftrightarrow & C_1.\mathtt{fd} \mapsto C_2 \\
C_1 \overset{S}{\curvearrowright} C_2 & \Leftrightarrow & C_1.\mathtt{fd} \mapsto C_2 * \exists S'.\, C_1.size \overset{.5}{\mapsto} S' * S' \dot{\geq} S \\
C_1 \curvearrowleft C_2 & \Leftrightarrow & C_2.\mathtt{bk} \mapsto C_1 \\
C_1 \circlearrowright C_2 & \Leftrightarrow & C_1 \curvearrowright C_2 * C_1 \curvearrowleft C_2 \\
C_1 \overset{S}{\circlearrowright} C_2 & \Leftrightarrow & C_1 \overset{S}{\curvearrowright} C_2 * C_1 \curvearrowleft C_2 \\
C_1 \circlearrowright^* C_2 & \Leftrightarrow & C_1 \doteq C_2 \vee \exists C'.\, C_1 \circlearrowright C' * C' \circlearrowright^* C_2 \\
C_1 \overset{S}{\circlearrowright}{}^* C_2 & \Leftrightarrow & C_1 \doteq C_2 \vee \exists C'.\, C_1 \overset{S}{\circlearrowright} C' * C' \overset{S}{\circlearrowright}{}^* C_2
\end{array}
$$

We begin by tackling a simplified scenario, in which we ignore treebins. Assume chunks have sizes that are multiples of 8 bytes up to 256 bytes (exclusive). All free chunks are thus contained in one of 32 smallbins. We shall also ignore the smallmap for now.

$$
\begin{array}{rcl}
node_S(C) & \Leftrightarrow & busynode_S(C) \vee freenode_S(C) \\
busynode_S(C) & \Leftrightarrow & \exists S', C_n. \\
 & & * \; C.cinuse \mapsto 1 \\
 & & * \; C.size \mapsto S' * S' \dot{\geq} S \\
 & & * \; C_n \doteq C + S' \\
 & & * \; node_0(C_n) \\
 & & * \; C_n.pinuse \mapsto 1 \\
freenode_S(C) & \Leftrightarrow & \exists S', C_n. \\
 & & * \; C.cinuse \mapsto 0 \\
 & & * \; C.size \overset{.5}{\mapsto} S' * S' \dot{\geq} S \\
 & & * \; snode_S(C) \\
 & & * \; C_n \doteq C + S' \\
 & & * \; busynode_0(C_n) \\
 & & * \; C_n.prevfoot \mapsto S' \\
 & & * \; C_n.pinuse \mapsto 0 \\
snode_S(C) & \Leftrightarrow & \exists C' \neq C.\, C \overset{S}{\circlearrowright} C' * C' \overset{S}{\circlearrowright}{}^* C \\
sbinheader_S(B) & \Leftrightarrow & B \circlearrowright B \vee \exists C \neq B.\, B \circlearrowright C * C \overset{S}{\circlearrowright}{}^* B
\end{array}
$$

If `start` is the address of the first chunk in the arena, then the entire state can be described by:

$$node_0(\mathtt{start}) * \forall^* i \in [2, 32).\, sbin_{8i}(\mathtt{smallbins} + 2i)$$

TODO:

- make a special case for the first node (its PINUSE flag should be permanently set)

Chapter 3

Operations on smallbins and trees

Then let

$$
\begin{aligned}
\mathit{sbinempty}(B) &\Leftrightarrow B \circlearrowleft B \\
\mathit{sbinnonempty}(S,B) &\Leftrightarrow \exists C {\neq} B.\, B \circlearrowleft C * C \overset{S}{\circlearrowleft}{}^{*} B \\
\mathit{sbin}(S,B) &\Leftrightarrow \mathit{sbinempty}(B) \vee \mathit{sbinnonempty}(S,B)
\end{aligned}
$$

The following predicate describes a valid `smallbins` array and corresponding `smallmap` vector.

$$
\begin{aligned}
\mathit{sbinAndMap}(i) &\Leftrightarrow \exists S,B.\ \ i \dot{\in} [2,32) * S \dot{=} 8i * B \dot{=} \texttt{smallbins}{+}2i * \\
&\qquad\quad (\texttt{\&smallmap}_i \mapsto 0 * \mathit{sbinempty}(B) \\
&\qquad\quad \vee \texttt{\&smallmap}_i \mapsto 1 * \mathit{sbinnonempty}(S,B)) \\
\mathit{sbinsOK} &\Leftrightarrow \forall^{*} i \in [2,32).\, \mathit{sbinAndMap}(i)
\end{aligned}
$$

The `insert_small_chunk` macro can be specified like so. Note that it preserves the *sbinsOK* predicate:

$$
\left\{\exists i.\, i \dot{\in} [2,32) * \texttt{S} \dot{=} 8i * \texttt{P} \overset{\texttt{S}}{\rightsquigarrow} \_ * \_ \leftsquigarrow \texttt{P} * \mathit{sbinsOK}\right\} \texttt{insert\_small\_chunk(M,P,S)} \left\{\mathit{sbinsOK}\right\}
$$

The details are as follows.

$$\left\{\exists i.\, i \dot{\in} [2,32) * \texttt{S} \dot{=} 8i * \texttt{P} \overset{\texttt{S}}{\rightsquigarrow} \_ * \_ \leftsquigarrow \texttt{P} * \mathit{sbinsOK}\right\}$$

```
bindex_t I = small_index(S);
```

$$\left\{\texttt{P} \overset{\texttt{S}}{\rightsquigarrow} \_ * \_ \leftsquigarrow \texttt{P} * \mathit{sbinsOK} * \texttt{I} \dot{\in} [2,32) * \texttt{S} \dot{=} 8\texttt{I}\right\}$$

$$\left\{\texttt{P} \overset{\texttt{S}}{\rightsquigarrow} \_ * \_ \leftsquigarrow \texttt{P} * (\mathit{sbinAndMap}(\texttt{I}) \mathrel{-\!\!*} \mathit{sbinsOK}) * \mathit{sbinAndMap}(\texttt{I}) * \texttt{I} \dot{\in} [2,32) * \texttt{S} \dot{=} 8\texttt{I}\right\}$$

```
mchunkptr B = smallbin_at(M, I);
```

$$\left\{\texttt{P} \overset{\texttt{S}}{\rightsquigarrow} \_ * \_ \leftsquigarrow \texttt{P} * (\mathit{sbinAndMap}(\texttt{I}) \mathrel{-\!\!*} \mathit{sbinsOK}) * \mathit{sbinAndMap}(\texttt{I}) * \texttt{I} \dot{\in} [2,32) * \texttt{S} \dot{=} 8\texttt{I} * \texttt{B} \dot{=} \texttt{smallbins+2I}\right\}$$

```
mchunkptr F = B;
assert(S >= MIN_CHUNK_SIZE);
```

$$\left\{\begin{array}{l}\texttt{P} \overset{\texttt{S}}{\rightsquigarrow} \_ * \_ \leftsquigarrow \texttt{P} * (\mathit{sbinAndMap}(\texttt{I}) \mathrel{-\!\!*} \mathit{sbinsOK}) * \mathit{sbinAndMap}(\texttt{I}) \\ * \texttt{I} \dot{\in} [2,32) * \texttt{S} \dot{=} 8\texttt{I} * \texttt{B} \dot{=} \texttt{smallbins+2I} * \texttt{F} \dot{=} \texttt{B}\end{array}\right\}$$

```
if (!smallmap_is_marked(M, I))
```

$$\left\{\begin{array}{l}\texttt{P} \overset{\texttt{S}}{\rightsquigarrow} \_ * \_ \leftsquigarrow \texttt{P} * (\mathit{sbinAndMap}(\texttt{I}) \mathrel{-\!\!*} \mathit{sbinsOK}) * \texttt{\&smallmap}_{\texttt{I}} \mapsto 0 * \mathit{sbinempty}(\texttt{B}) \\ * \texttt{I} \dot{\in} [2,32) * \texttt{S} \dot{=} 8\texttt{I} * \texttt{B} \dot{=} \texttt{smallbins+2I} * \texttt{F} \dot{=} \texttt{B}\end{array}\right\}$$

```
  mark_smallmap(M, I);
```

$$\left\{\begin{array}{l}\texttt{P} \overset{\texttt{S}}{\rightsquigarrow} \_ * \_ \leftsquigarrow \texttt{P} * (\mathit{sbinAndMap}(\texttt{I}) \mathrel{-\!\!*} \mathit{sbinsOK}) * \texttt{\&smallmap}_{\texttt{I}} \mapsto 1 * \mathit{sbinempty}(\texttt{B}) \\ * \texttt{I} \dot{\in} [2,32) * \texttt{S} \dot{=} 8\texttt{I} * \texttt{B} \dot{=} \texttt{smallbins+2I} * \texttt{F} \dot{=} \texttt{B}\end{array}\right\}$$

$$\left\{\begin{array}{l}\mathtt{P}\overset{\mathtt{S}}{\rightharpoondown}\_ \ast \_\leftharpoondown\mathtt{P} \ast (\mathit{sbinAndMap}(\mathtt{I}) \mathrel{-\!\!\ast} \mathit{sbinsOK}) \ast \&\mathtt{smallmap}_{\mathtt{I}} \mapsto 1 \ast \mathtt{B}\circlearrowright\mathtt{F} \\ \ast\ \mathtt{I}\,\dot{\in}[2,32) \ast \mathtt{S}\doteq 8\mathtt{I} \ast \mathtt{B}\doteq\mathtt{smallbins}{+}2\mathtt{I} \ast \mathtt{F}\doteq\mathtt{B} \ast \mathtt{B}\dot{\neq}\mathtt{P}\end{array}\right\}$$

`else if (RTCHECK(ok_address(M, B->fd)))`

$$\left\{\begin{array}{l}\mathtt{P}\overset{\mathtt{S}}{\rightharpoondown}\_ \ast \_\leftharpoondown\mathtt{P} \ast (\mathit{sbinAndMap}(\mathtt{I}) \mathrel{-\!\!\ast} \mathit{sbinsOK}) \ast \&\mathtt{smallmap}_{\mathtt{I}} \mapsto 1 \ast \mathit{sbinnonempty}(\mathtt{S},\mathtt{B}) \\ \ast\ \mathtt{I}\,\dot{\in}[2,32) \ast \mathtt{S}\doteq 8\mathtt{I} \ast \mathtt{B}\doteq\mathtt{smallbins}{+}2\mathtt{I}\end{array}\right\}$$

$$\left\{\begin{array}{l}\exists C\dot{\neq}\mathtt{B}.\,\mathtt{P}\overset{\mathtt{S}}{\rightharpoondown}\_ \ast \_\leftharpoondown\mathtt{P} \ast (\mathit{sbinAndMap}(\mathtt{I}) \mathrel{-\!\!\ast} \mathit{sbinsOK}) \ast \&\mathtt{smallmap}_{\mathtt{I}} \mapsto 1 \ast \mathtt{B}\circlearrowright C \ast C\overset{\mathtt{S}}{\circlearrowright}{}^{*}\mathtt{B} \\ \ast\ \mathtt{I}\,\dot{\in}[2,32) \ast \mathtt{S}\doteq 8\mathtt{I} \ast \mathtt{B}\doteq\mathtt{smallbins}{+}2\mathtt{I} \ast \mathtt{B}\dot{\neq}\mathtt{P}\end{array}\right\}$$

`F = B->fd;`

$$\left\{\begin{array}{l}\mathtt{P}\overset{\mathtt{S}}{\rightharpoondown}\_ \ast \_\leftharpoondown\mathtt{P} \ast (\mathit{sbinAndMap}(\mathtt{I}) \mathrel{-\!\!\ast} \mathit{sbinsOK}) \ast \&\mathtt{smallmap}_{\mathtt{I}} \mapsto 1 \ast \mathtt{B}\circlearrowright\mathtt{F} \ast \mathtt{F}\overset{\mathtt{S}}{\circlearrowright}{}^{*}\mathtt{B} \\ \ast\ \mathtt{I}\,\dot{\in}[2,32) \ast \mathtt{S}\doteq 8\mathtt{I} \ast \mathtt{B}\doteq\mathtt{smallbins}{+}2\mathtt{I} \ast \mathtt{B}\dot{\neq}\mathtt{P} \ast \mathtt{F}\dot{\neq}\mathtt{B}\end{array}\right\}$$

```
else {
  CORRUPTION_ERROR_ACTION(M);
}
```

$$\left\{\begin{array}{l}\mathtt{P}\overset{\mathtt{S}}{\rightharpoondown}\_ \ast \_\leftharpoondown\mathtt{P} \ast (\mathit{sbinAndMap}(\mathtt{I}) \mathrel{-\!\!\ast} \mathit{sbinsOK}) \ast \&\mathtt{smallmap}_{\mathtt{I}} \mapsto 1 \ast \mathtt{B}\circlearrowright\mathtt{F} \ast \mathtt{F}\circlearrowright^{*}\mathtt{B} \\ \ast\ \mathtt{I}\,\dot{\in}[2,32) \ast \mathtt{S}\doteq 8\mathtt{I} \ast \mathtt{B}\doteq\mathtt{smallbins}{+}2\mathtt{I} \ast \mathtt{B}\dot{\neq}\mathtt{P}\end{array}\right\}$$

`B->fd = P;`

$$\left\{\begin{array}{l}\mathtt{P}\overset{\mathtt{S}}{\rightharpoondown}\_ \ast \_\leftharpoondown\mathtt{P} \ast (\mathit{sbinAndMap}(\mathtt{I}) \mathrel{-\!\!\ast} \mathit{sbinsOK}) \ast \&\mathtt{smallmap}_{\mathtt{I}} \mapsto 1 \ast \mathtt{B}\rightharpoondown\mathtt{P} \ast \mathtt{B}\leftharpoondown\mathtt{F} \ast \mathtt{F}\circlearrowright^{*}\mathtt{B} \\ \ast\ \mathtt{I}\,\dot{\in}[2,32) \ast \mathtt{S}\doteq 8\mathtt{I} \ast \mathtt{B}\doteq\mathtt{smallbins}{+}2\mathtt{I} \ast \mathtt{B}\dot{\neq}\mathtt{P}\end{array}\right\}$$

`F->bk = P;`

$$\left\{\begin{array}{l}\mathtt{P}\overset{\mathtt{S}}{\rightharpoondown}\_ \ast \_\leftharpoondown\mathtt{P} \ast (\mathit{sbinAndMap}(\mathtt{I}) \mathrel{-\!\!\ast} \mathit{sbinsOK}) \ast \&\mathtt{smallmap}_{\mathtt{I}} \mapsto 1 \ast \mathtt{B}\rightharpoondown\mathtt{P} \ast \mathtt{P}\leftharpoondown\mathtt{F} \ast \mathtt{F}\circlearrowright^{*}\mathtt{B} \\ \ast\ \mathtt{I}\,\dot{\in}[2,32) \ast \mathtt{S}\doteq 8\mathtt{I} \ast \mathtt{B}\doteq\mathtt{smallbins}{+}2\mathtt{I} \ast \mathtt{B}\dot{\neq}\mathtt{P}\end{array}\right\}$$

`P->fd = F;`

$$\left\{\begin{array}{l}\_\leftharpoondown\mathtt{P} \ast (\mathit{sbinAndMap}(\mathtt{I}) \mathrel{-\!\!\ast} \mathit{sbinsOK}) \ast \&\mathtt{smallmap}_{\mathtt{I}} \mapsto 1 \ast \mathtt{B}\rightharpoondown\mathtt{P} \ast \mathtt{P}\overset{S}{\circlearrowright}\mathtt{F} \ast \mathtt{F}\circlearrowright^{*}\mathtt{B} \\ \ast\ \mathtt{I}\,\dot{\in}[2,32) \ast \mathtt{S}\doteq 8\mathtt{I} \ast \mathtt{B}\doteq\mathtt{smallbins}{+}2\mathtt{I} \ast \mathtt{B}\dot{\neq}\mathtt{P}\end{array}\right\}$$

`P->bk = B;`

$$\left\{\begin{array}{l}(\mathit{sbinAndMap}(\mathtt{I}) \mathrel{-\!\!\ast} \mathit{sbinsOK}) \ast \&\mathtt{smallmap}_{\mathtt{I}} \mapsto 1 \ast \mathtt{B}\circlearrowright\mathtt{P} \ast \mathtt{P}\overset{S}{\circlearrowright}\mathtt{F} \ast \mathtt{F}\circlearrowright^{*}\mathtt{B} \\ \ast\ \mathtt{I}\,\dot{\in}[2,32) \ast \mathtt{S}\doteq 8\mathtt{I} \ast \mathtt{B}\doteq\mathtt{smallbins}{+}2\mathtt{I} \ast \mathtt{B}\dot{\neq}\mathtt{P}\end{array}\right\}$$

$$\left\{\begin{array}{l}(\mathit{sbinAndMap}(\mathtt{I}) \mathrel{-\!\!\ast} \mathit{sbinsOK}) \ast \&\mathtt{smallmap}_{\mathtt{I}} \mapsto 1 \ast \mathtt{B}\circlearrowright\mathtt{P} \ast \mathtt{P}\overset{S}{\circlearrowright}{}^{*}\mathtt{B} \\ \ast\ \mathtt{I}\,\dot{\in}[2,32) \ast \mathtt{S}\doteq 8\mathtt{I} \ast \mathtt{B}\doteq\mathtt{smallbins}{+}2\mathtt{I} \ast \mathtt{B}\dot{\neq}\mathtt{P}\end{array}\right\}$$

$$\left\{(\mathit{sbinAndMap}(\mathtt{I}) \mathrel{-\!\!\ast} \mathit{sbinsOK}) \ast \&\mathtt{smallmap}_{\mathtt{I}} \mapsto 1 \ast \mathit{sbinnonempty}(\mathtt{S},\mathtt{smallbins}{+}2\mathtt{I}) \ast \mathtt{I}\,\dot{\in}[2,32) \ast \mathtt{S}\doteq 8\mathtt{I}\right\}$$

$$\left\{(\mathit{sbinAndMap}(\mathtt{I}) \mathrel{-\!\!\ast} \mathit{sbinsOK}) \ast \mathit{sbinAndMap}(\mathtt{I})\right\}$$

$$\left\{\mathit{sbinsOK}\right\}$$

Chapter 4

tmalloc_small

Chapter 5

tmalloc_large

Chapter 6

sys_alloc

Chapter 7

dlmalloc

We now give the entire source code of `dlmalloc`, interspersed with an explanatory commentary and annotations for the safety proof. We shall prove that a successful call to `dlmalloc(mem)` allocates a chunk with payload no less that `mem` bytes. I think it's probably sufficient to provide only a lower bound on the payload size – it certainly simplifies the proof!

$\left\{p_1\right\}$

```
void* dlmalloc(size_t bytes) {
#if USE_LOCKS
  ensure_initialization(); /* initialize in sys_alloc if not using locks */
#endif
  if (!PREACTION(gm)) {
```

$\left\{-\right\}$

```
    void* mem;
    size_t nb;
    if (bytes <= MAX_SMALL_REQUEST) {
```

$\left\{P_{small}\right\}$ where $P_{small} = \texttt{bytes} \leq 244$

Allocating small chunks

```
      bindex_t idx;
      binmap_t smallbits;
      nb = (bytes < MIN_REQUEST)? MIN_CHUNK_SIZE : pad_request(bytes);
```

$\left\{P_{small} \wedge P_{nb}\right\}$ where $P_{nb} = \max\{16, \lceil \texttt{bytes} + 4 \rceil_8\}$

```
      idx = small_index(nb);
```

$\left\{P_{small} \wedge P_{nb} \wedge \texttt{idx} = \lfloor \texttt{nb}/8 \rfloor\right\}$

```
      smallbits = gm->smallmap >> idx;
```

$$\left\{\begin{array}{l} P_{small} \wedge P_{nb} \wedge \texttt{idx} = \lfloor \texttt{nb}/8 \rfloor \\ \wedge\ \forall i \in [0, 32 - \texttt{idx}).\ \texttt{smallbits}[i] = 0 \Leftrightarrow \mathit{smallbin}(i + \texttt{idx}) = \varnothing \end{array}\right\}$$

```
      if ((smallbits & 0x3U) != 0) { /* Remainderless fit to a smallbin. */
```

$\left\{P_{small} \wedge P_{nb} \wedge \texttt{idx} = \lfloor \texttt{nb}/8 \rfloor \wedge (\mathit{smallbin}(\texttt{idx}) \neq \varnothing \vee \mathit{smallbin}(\texttt{idx} + 1) \neq \varnothing)\right\}$

'Remainderless' fit to a smallbin

```
        mchunkptr b, p;
        idx += ~smallbits & 1; /* Uses next bin if idx empty */
```

$\left\{P_{small} \wedge P_{nb} \wedge (\texttt{idx} = \lfloor \texttt{nb}/8 \rfloor \vee \texttt{idx} = \lfloor \texttt{nb}/8 \rfloor + 1) \wedge smallbin(\texttt{idx}) \neq \varnothing\right\}$

```
b = smallbin_at(gm, idx);
```

$\left\{P_{small} \wedge P_{nb} \wedge (\texttt{idx} = \lfloor \texttt{nb}/8 \rfloor \vee \texttt{idx} = \lfloor \texttt{nb}/8 \rfloor + 1) \wedge \texttt{b} = smallbin(\texttt{idx}) \wedge \texttt{b} \neq \varnothing\right\}$

```
p = b->fd;
```

$$\left\{\begin{array}{l} P_{small} \wedge P_{nb} \wedge (\texttt{idx} = \lfloor \texttt{nb}/8 \rfloor \vee \texttt{idx} = \lfloor \texttt{nb}/8 \rfloor + 1) \\ \wedge\, \texttt{b} = smallbin(\texttt{idx}) \wedge \texttt{b} = \texttt{p} :: ps \wedge flags(\texttt{p}) = 01 \end{array}\right\}$$

```
assert(chunksize(p) == small_index2size(idx));
unlink_first_small_chunk(gm, b, p, idx);
```

$$\left\{\begin{array}{l} P_{small} \wedge P_{nb} \wedge (\texttt{idx} = \lfloor \texttt{nb}/8 \rfloor \vee \texttt{idx} = \lfloor \texttt{nb}/8 \rfloor + 1) \\ \wedge\, \texttt{b} = smallbin(\texttt{idx}) \wedge \texttt{b} = ps \\ \wedge\, flags(\texttt{p}) = \triangledown\blacktriangle \end{array}\right\}$$

```
set_inuse_and_pinuse(gm, p, small_index2size(idx));
```

$$\left\{\begin{array}{l} P_{small} \wedge P_{nb} \wedge (\texttt{idx} = \lfloor \texttt{nb}/8 \rfloor \vee \texttt{idx} = \lfloor \texttt{nb}/8 \rfloor + 1) \\ \wedge\, \texttt{b} = smallbin(\texttt{idx}) \wedge \texttt{b} = ps \\ \wedge\, flags(\texttt{p}) = \blacktriangledown\blacktriangle \wedge flags(next(\texttt{p})) = \blacktriangle \end{array}\right\}$$

```
mem = chunk2mem(p);
check_malloced_chunk(gm, mem, nb);
```

$$\left\{\begin{array}{l} P_{small} \wedge P_{nb} \wedge (\texttt{idx} = \lfloor \texttt{nb}/8 \rfloor \vee \texttt{idx} = \lfloor \texttt{nb}/8 \rfloor + 1) \\ \wedge\, \texttt{b} = smallbin(\texttt{idx}) \wedge \texttt{b} = ps \\ \wedge\, flags(\texttt{p}) = \blacktriangledown\blacktriangle \wedge flags(next(\texttt{p})) = \blacktriangle \wedge \texttt{mem} = \texttt{p} - 2 \end{array}\right\}$$

```
    goto postaction;
  }
  else if (nb > gm->dvsize) {
```

$$\left\{\begin{array}{l} P_{small} \wedge P_{nb} \wedge \texttt{idx} = \lfloor \texttt{nb}/8 \rfloor \\ \wedge\, \forall i \in [0, 32 - \texttt{idx}).\, \texttt{smallbits}[i] = 0 \Leftrightarrow smallbin(i + \texttt{idx}) = \varnothing \\ \wedge\, smallbin(\texttt{idx}) = smallbin(\texttt{idx} + 1) = \varnothing \end{array}\right\}$$

```
    if (smallbits != 0) { /* Use chunk in next nonempty smallbin */
```

$$\left\{\begin{array}{l} P_{small} \wedge P_{nb} \wedge \texttt{idx} = \lfloor \texttt{nb}/8 \rfloor \\ \wedge\, \forall i \in [0, 32 - \texttt{idx}).\, \texttt{smallbits}[i] = 0 \Leftrightarrow smallbin(i + \texttt{idx}) = \varnothing \\ \wedge\, \exists i \in [\texttt{idx} + 2, 32).\, smallbin(i) \neq \varnothing \end{array}\right\}$$

'Remainderful' fit to a smallbin

```
mchunkptr b, p, r;
size_t rsize;
bindex_t i;
binmap_t leftbits = (smallbits << idx) & left_bits(idx2bit(idx));
```

$$\left\{\begin{array}{l} P_{small} \wedge P_{nb} \wedge \texttt{idx} = \lfloor \texttt{nb}/8 \rfloor \\ \wedge\, \forall i \in [\texttt{idx} + 1, 32).\, \texttt{leftbits}[i] = 0 \Leftrightarrow smallbin(i) = \varnothing \\ \wedge\, \exists i \in [\texttt{idx} + 2, 32).\, smallbin(i) \neq \varnothing \end{array}\right\}$$

```
binmap_t leastbit = least_bit(leftbits);
```

$$\left\{\begin{array}{l} P_{small} \wedge P_{nb} \wedge \texttt{idx} = \lfloor \texttt{nb}/8 \rfloor \\ \wedge\, \forall i \in [\texttt{idx} + 1, 32).\, \texttt{leftbits}[i] = 0 \Leftrightarrow smallbin(i) = \varnothing \\ \wedge\, \lfloor \log_2(\texttt{leastbit}) \rfloor = \min\{i \in [\texttt{idx} + 2, 32) \mid smallbin(i) \neq \varnothing\} \end{array}\right\}$$

```
compute_bit2idx(leastbit, i);
```

$$\left\{\begin{array}{l} P_{small} \wedge P_{nb} \wedge \texttt{idx} = \lfloor \texttt{nb}/8 \rfloor \\ \wedge\ \forall i \in [\texttt{idx}+1, 32).\ \texttt{leftbits}[i] = 0 \Leftrightarrow smallbin(i) = \varnothing \\ \wedge\ \texttt{i} = \min\{i \in [\texttt{idx}+2, 32) \mid smallbin(i) \neq \varnothing\} \end{array}\right\}$$

$$\left\{P_{small} \wedge P_{nb} \wedge \texttt{i} \geq \lfloor \texttt{nb}/8 \rfloor + 2 \wedge smallbin(\texttt{i}) \neq \varnothing\right\}$$

```
b = smallbin_at(gm, i);
```

$$\left\{P_{small} \wedge P_{nb} \wedge \texttt{i} \geq \lfloor \texttt{nb}/8 \rfloor + 2 \wedge \texttt{b} = smallbin(\texttt{i}) \wedge \texttt{b} \neq \varnothing\right\}$$

```
p = b->fd;
```

$$\left\{P_{small} \wedge P_{nb} \wedge \texttt{i} \geq \lfloor \texttt{nb}/8 \rfloor + 2 \wedge \texttt{b} = smallbin(\texttt{i}) \wedge \texttt{b} = \texttt{p} :: ps\right\}$$

```
assert(chunksize(p) == small_index2size(i));
unlink_first_small_chunk(gm, b, p, i);
```

$$\left\{\begin{array}{l} P_{small} \wedge P_{nb} \wedge \texttt{i} \geq \lfloor \texttt{nb}/8 \rfloor + 2 \wedge \texttt{b} = smallbin(\texttt{i}) \wedge \texttt{b} = ps \\ \wedge\ flags(\texttt{p}) = \triangledown\blacktriangle \end{array}\right\}$$

```
rsize = small_index2size(i) - nb;
```

$$\left\{P_{small} \wedge P_{nb} \wedge flags(\texttt{p}) = \triangledown\blacktriangle \wedge \texttt{rsize} = size(p) - \texttt{nb}\right\}$$

```
/* Fit here cannot be remainderless if 4byte sizes */
if (SIZE_T_SIZE != 4 && rsize < MIN_CHUNK_SIZE)
```

$$\left\{false\right\}$$

```
  set_inuse_and_pinuse(gm, p, small_index2size(i));
else {
  set_size_and_pinuse_of_inuse_chunk(gm, p, nb);
```

$$\left\{P_{small} \wedge P_{nb} \wedge flags(\texttt{p}) = \blacktriangledown\blacktriangle \wedge size(p) = \texttt{nb} \wedge \texttt{rsize} = size(p) - \texttt{nb}\right\}$$

```
  r = chunk_plus_offset(p, nb);
```

$$\left\{\begin{array}{l} P_{small} \wedge P_{nb} \wedge flags(\texttt{p}) = \blacktriangledown\blacktriangle \wedge size(p) = \texttt{nb} \wedge \texttt{rsize} = size(p) - \texttt{nb} \\ \wedge\ \texttt{r} = \texttt{p} + \texttt{nb} \end{array}\right\}$$

```
  set_size_and_pinuse_of_free_chunk(r, rsize);
```

$$\left\{\begin{array}{l} P_{small} \wedge P_{nb} \wedge flags(\texttt{p}) = \blacktriangledown\blacktriangle \wedge size(p) = \texttt{nb} \wedge \texttt{rsize} = size(p) - \texttt{nb} \\ \wedge\ \texttt{r} = \texttt{p} + \texttt{nb} \wedge flags(\texttt{r}) = \triangledown\blacktriangle \wedge size(\texttt{r}) = \texttt{rsize} \end{array}\right\}$$

```
  replace_dv(gm, r, rsize);
```

$$\left\{\begin{array}{l} P_{small} \wedge P_{nb} \wedge flags(\texttt{p}) = \blacktriangledown\blacktriangle \wedge size(p) = \texttt{nb} \wedge \texttt{rsize} = size(p) - \texttt{nb} \\ \wedge\ \texttt{r} = \texttt{p} + \texttt{nb} \wedge flags(\texttt{r}) = \triangledown\blacktriangle \wedge size(\texttt{r}) = \texttt{rsize} \\ \wedge\ \texttt{dv} = \texttt{r} \wedge \texttt{dvsize} = \texttt{rsize} \end{array}\right\}$$

```
}
mem = chunk2mem(p);
check_malloced_chunk(gm, mem, nb);
```

$$\left\{\begin{array}{l} P_{small} \wedge P_{nb} \wedge flags(\texttt{p}) = \blacktriangledown\blacktriangle \wedge size(p) = \texttt{nb} \wedge \texttt{rsize} = size(p) - \texttt{nb} \\ \wedge\ \texttt{r} = \texttt{p} + \texttt{nb} \wedge flags(\texttt{r}) = \triangledown\blacktriangle \wedge size(\texttt{r}) = \texttt{rsize} \wedge \texttt{mem} = \texttt{p} + 2 \end{array}\right\}$$

```
  goto postaction;
}
```

Using a treebin instead

```
else if (gm->treemap != 0 && (mem = tmalloc_small(gm, nb)) != 0) {
```

$$\left\{\begin{array}{l} P_{small} \wedge P_{nb} \wedge \texttt{idx} = \lfloor \texttt{nb}/8 \rfloor \wedge \forall i \in [\texttt{idx}, 32).\ smallbin(i) = \varnothing \\ \wedge\ \texttt{mem} = p + 2 \wedge flags(p) = \blacktriangledown\blacktriangle \wedge size(p) \geq \texttt{nb} \end{array}\right\}$$

```
  check_malloced_chunk(gm, mem, nb);
  goto postaction;
```

```
      }
    }
  }
```

Allocating large chunks

```
  else if (bytes >= MAX_REQUEST)
```
$$\{\texttt{bytes} \geq 2^{32} - 63\}$$
```
    nb = MAX_SIZE_T; /* Too big to allocate. Force failure (in sys alloc) */
```
$$\{\texttt{nb} = 2^{32} - 1\}$$
```
  else {
```
$$\{P_{large}\} \text{ where } P_{large} = 244 < \texttt{bytes} < 2^{32} - 63$$
```
    nb = pad_request(bytes);
```
$$\{P_{large} \wedge P_{nb}\}$$
```
    if (gm->treemap != 0 && (mem = tmalloc_large(gm, nb)) != 0) {
```
$$\{P_{large} \wedge P_{nb} \wedge \texttt{mem} = p + 2 \wedge \mathit{flags}(p) = \blacktriangledown\blacktriangle \wedge \mathit{size}(p) \geq \texttt{nb}\}$$
```
      check_malloced_chunk(gm, mem, nb);
      goto postaction;
    }
  }
```

Using the designated victim

$$\{P_{nb}\}$$
```
  if (nb <= gm->dvsize) {
```
$$\{P_{nb} \wedge \texttt{nb} \leq \texttt{dvsize}\}$$
```
    size_t rsize = gm->dvsize - nb;
```
$$\{P_{nb} \wedge \texttt{nb} \leq \texttt{dvsize} \wedge \texttt{rsize} = \texttt{dvsize} - \texttt{nb}\}$$
```
    mchunkptr p = gm->dv;
```
$$\{P_{nb} \wedge \texttt{nb} \leq \mathit{size}(\texttt{p}) \wedge \texttt{rsize} = \mathit{size}(\texttt{p}) - \texttt{nb} \wedge \mathit{flags}(\texttt{p}) = \triangledown\blacktriangle\}$$
```
    if (rsize >= MIN_CHUNK_SIZE) { /* split dv */
```
$$\{P_{nb} \wedge \texttt{rsize} = \mathit{size}(\texttt{p}) - \texttt{nb} \wedge \texttt{rsize} \geq 16 \wedge \mathit{flags}(\texttt{p}) = \triangledown\blacktriangle\}$$
```
      mchunkptr r = gm->dv = chunk_plus_offset(p, nb);
```
$$\{P_{nb} \wedge \texttt{rsize} = \mathit{size}(\texttt{p}) - \texttt{nb} \wedge \texttt{rsize} \geq 16 \wedge \texttt{r} = \texttt{p} + \texttt{nb} \wedge \mathit{flags}(\texttt{p}) = \triangledown\blacktriangle\}$$
```
      gm->dvsize = rsize;
      set_size_and_pinuse_of_free_chunk(r, rsize);
```
$$\left\{\begin{array}{l} P_{nb} \wedge \texttt{rsize} = \mathit{size}(\texttt{p}) - \texttt{nb} \wedge \texttt{rsize} \geq 16 \wedge \texttt{r} = \texttt{p} + \texttt{nb} \wedge \mathit{flags}(\texttt{p}) = \triangledown\blacktriangle \\ \wedge\, \mathit{flags}(\texttt{r}) = \triangledown\blacktriangle \wedge \mathit{size}(\texttt{r}) = \texttt{rsize} \end{array}\right\}$$
```
      set_size_and_pinuse_of_inuse_chunk(gm, p, nb);
```
$$\left\{\begin{array}{l} P_{nb} \wedge \texttt{rsize} \geq 16 \wedge \texttt{r} = \texttt{p} + \texttt{nb} \wedge \mathit{flags}(\texttt{p}) = \blacktriangledown\blacktriangle \wedge \mathit{size}(p) = \texttt{nb} \\ \wedge\, \mathit{flags}(\texttt{r}) = \triangledown\blacktriangle \wedge \mathit{size}(\texttt{r}) = \texttt{rsize} \end{array}\right\}$$
```
    }
    else { /* exhaust dv */
```
$$\{P_{nb} \wedge (\mathit{size}(\texttt{p}) = \texttt{nb} \vee \mathit{size}(\texttt{p}) = \texttt{nb} + 8) \wedge \mathit{flags}(\texttt{p}) = \triangledown\blacktriangle\}$$
```
      size_t dvs = gm->dvsize;
      gm->dvsize = 0;
      gm->dv = 0;
```

```
      set_inuse_and_pinuse(gm, p, dvs);
```

$$\left\{P_{nb} \wedge (\mathit{size}(\mathtt{p}) = \mathtt{nb} \vee \mathit{size}(\mathtt{p}) = \mathtt{nb} + 8) \wedge \mathit{flags}(\mathtt{p}) = \blacktriangledown\blacktriangle\right\}$$

```
    }
```

$$\left\{P_{nb} \wedge (\mathit{size}(\mathtt{p}) = \mathtt{nb} \vee \mathit{size}(\mathtt{p}) = \mathtt{nb} + 8) \wedge \mathit{flags}(\mathtt{p}) = \blacktriangledown\blacktriangle\right\}$$

```
    mem = chunk2mem(p);
    check_malloced_chunk(gm, mem, nb);
```

$$\left\{P_{nb} \wedge (\mathit{size}(\mathtt{p}) = \mathtt{nb} \vee \mathit{size}(\mathtt{p}) = \mathtt{nb} + 8) \wedge \mathit{flags}(\mathtt{p}) = \blacktriangledown\blacktriangle \wedge \mathtt{mem} = \mathtt{p} + 2\right\}$$

```
    goto postaction;
  }
```

Using the top chunk

```
  else if (nb < gm->topsize) { /* Split top */
```

$$\left\{P_{nb} \wedge \mathtt{nb} < \mathit{size}(\mathtt{top})\right\}$$

```
    size_t rsize = gm->topsize -= nb;
```

$$\left\{P_{nb} \wedge \mathtt{rsize} = \mathit{size}(\mathtt{top}) - \mathtt{nb} \wedge \mathtt{rsize} > 0\right\}$$

```
    mchunkptr p = gm->top;
```

$$\left\{P_{nb} \wedge \mathtt{rsize} = \mathit{size}(\mathtt{p}) - \mathtt{nb} \wedge \mathtt{rsize} > 0\right\}$$

```
    mchunkptr r = gm->top = chunk_plus_offset(p, nb);
```

$$\left\{P_{nb} \wedge \mathtt{rsize} = \mathit{size}(\mathtt{p}) - \mathtt{nb} \wedge \mathtt{rsize} > 0 \wedge \mathtt{r} = \mathtt{p} + \mathtt{nb}\right\}$$

```
    r->head = rsize | PINUSE_BIT;
```

$$\left\{P_{nb} \wedge \mathit{size}(\mathtt{r}) = \mathit{size}(\mathtt{p}) - \mathtt{nb} \wedge \mathit{size}(\mathtt{r}) > 0 \wedge \mathit{flags}(\mathtt{r}) = \triangledown\blacktriangle \wedge \mathtt{r} = \mathtt{p} + \mathtt{nb}\right\}$$

```
    set_size_and_pinuse_of_inuse_chunk(gm, p, nb);
```

$$\left\{P_{nb} \wedge \mathit{size}(\mathtt{p}) = \mathtt{nb} \wedge \mathit{flags}(\mathtt{p}) = \blacktriangledown\blacktriangle \wedge \mathit{size}(\mathtt{r}) > 0 \wedge \mathit{flags}(\mathtt{r}) = \triangledown\blacktriangle \wedge \mathtt{r} = \mathtt{p} + \mathtt{nb}\right\}$$

```
    mem = chunk2mem(p);
```

$$\left\{P_{nb} \wedge \mathit{size}(\mathtt{p}) = \mathtt{nb} \wedge \mathit{flags}(\mathtt{p}) = \blacktriangledown\blacktriangle \wedge \mathtt{mem} = \mathtt{p} + 2\right\}$$

```
    check_top_chunk(gm, gm->top);
    check_malloced_chunk(gm, mem, nb);
    goto postaction;
  }
```

Obtaining memory from the system

```
    mem = sys_alloc(gm, nb);
  postaction:
    POSTACTION(gm);
    return mem;
  }
  return 0;
}
```

Chapter 8

dlfree

```
void dlfree(void* mem) {
  if (mem != 0) {
    mchunkptr p = mem2chunk(mem);
#if FOOTERS
    mstate fm = get_mstate_for(p);
    if (!ok_magic(fm)) {
      USAGE_ERROR_ACTION(fm, p);
      return;
    }
#else /* FOOTERS */
#define fm gm
#endif /* FOOTERS */
    if (!PREACTION(fm)) {
      check_inuse_chunk(fm, p);
      if (RTCHECK(ok_address(fm, p) && ok_inuse(p))) {
        size_t psize = chunksize(p);
        mchunkptr next = chunk_plus_offset(p, psize);
        if (!pinuse(p)) {
          size_t prevsize = p->prev_foot;
          if (is_mmapped(p)) {
            psize += prevsize + MMAP_FOOT_PAD;
            if (CALL_MUNMAP((char*)p - prevsize, psize) == 0)
              fm->footprint -= psize;
            goto postaction;
          }
          else {
            mchunkptr prev = chunk_minus_offset(p, prevsize);
            psize += prevsize;
            p = prev;
            if (RTCHECK(ok_address(fm, prev))) { /* consolidate backward */
              if (p != fm->dv) {
                unlink_chunk(fm, p, prevsize);
              }
              else if ((next->head & INUSE_BITS) == INUSE_BITS) {
                fm->dvsize = psize;
                set_free_with_pinuse(p, psize, next);
                goto postaction;
              }
```

```
      }
      else
        goto erroraction;
    }
  }

  if (RTCHECK(ok_next(p, next) && ok_pinuse(next))) {
    if (!cinuse(next)) { /* consolidate forward */
      if (next == fm->top) {
        size_t tsize = fm->topsize += psize;
        fm->top = p;
        p->head = tsize | PINUSE_BIT;
        if (p == fm->dv) {
          fm->dv = 0;
          fm->dvsize = 0;
        }
        if (should_trim(fm, tsize))
          sys_trim(fm, 0);
        goto postaction;
      }
      else if (next == fm->dv) {
        size_t dsize = fm->dvsize += psize;
        fm->dv = p;
        set_size_and_pinuse_of_free_chunk(p, dsize);
        goto postaction;
      }
      else {
        size_t nsize = chunksize(next);
        psize += nsize;
        unlink_chunk(fm, next, nsize);
        set_size_and_pinuse_of_free_chunk(p, psize);
        if (p == fm->dv) {
          fm->dvsize = psize;
          goto postaction;
        }
      }
    }
    else
      set_free_with_pinuse(p, psize, next);

    if (is_small(psize)) {
      insert_small_chunk(fm, p, psize);
      check_free_chunk(fm, p);
    }
    else {
      tchunkptr tp = (tchunkptr)p;
      insert_large_chunk(fm, tp, psize);
      check_free_chunk(fm, p);
      if (--fm->release_checks == 0)
        release_unused_segments(fm);
    }
    goto postaction;
  }
}
```

```
    erroraction:
      USAGE_ERROR_ACTION(fm, p);
    postaction:
      POSTACTION(fm);
    }
  }
#if !FOOTERS
#undef fm
#endif /* FOOTERS */
}
```